BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

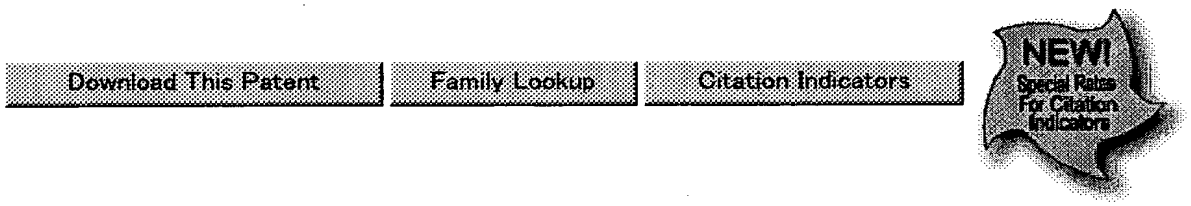
As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

☐ Includ

MicroPatent® PatSearch FullText: Record 1 of 1

Search scope: USG USA EPA EPB WO JP; Full patent spec.
Years: 1971–2001
Text: Patent/Publication No.: JP08249183

[no drawing available]



Go to first matching text

JP08249183 A2
EXECUTION FOR INFERENCE PARALLEL INSTRUCTION THREADS
INTERNATL BUSINESS MACH CORP <IBM>
Inventor(s): DUBEY PRADEEP KUMAR ;BARTON CHARLES MARSHALL ;CHUANG CHIAO-MEI ;LAM LINH HUE ;O'BRIAN JOHN KEVIN ;O'BRIAN KATHRYN MARY
Application No. 08011080 JP08011080 JP, Filed 19960125, Published 19960927

Abstract: PROBLEM TO BE SOLVED: To provide a central processing unit(CPU) inside a computer for making the inference parallel execution of plural instruction threads possible.

SOLUTION: This CPU uses a fork–interruption instruction added to the instruction set of the CPU and inserted in a program before execution time so as to indicate a latent future thread for parallel execution. The CPU is provided with an instruction cache provided with one or plural instruction cache ports, a bank composed of one or plural program counters, the bank composed of one or plural dispatchers, a thread management unit for processing inter– thread communication and abandoning the future thread violating dependency relation, a set of architectured registers in common to all the threads and a scheduler for scheduling the parallel execution of the instructions on one or plural function units inside the CPU.

Int'l Class: G06F00938;

Priority: US 95 383331 19950203

MicroPatent Reference Number: 000244344

☐ Includ

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号

特開平8－249183

(43)公開日　平成８年(1996)９月27日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ　9/38	３７０		Ｇ０６Ｆ　9/38	３７０Ａ	

審査請求　未請求　請求項の数38　ＯＬ　（全 32 頁）

(21)出願番号　特願平8－11080

(22)出願日　平成８年(1996)１月25日

(31)優先権主張番号　３８３３３１

(32)優先日　1995年２月３日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　390009531
インターナショナル・ビジネス・マシーンズ・コーポレイション
ＩＮＴＥＲＮＡＴＩＯＮＡＬ　ＢＵＳＩＮＥＳＳ　ＭＡＳＣＨＩＮＥＳ　ＣＯＲＰＯＲＡＴＩＯＮ
アメリカ合衆国10504、ニューヨーク州　アーモンク　（番地なし）

(72)発明者　プラデープ・クマル・ダビー
アメリカ合衆国10606　ニューヨーク州ホワイト・プレインズ　マーティン・アベニュー　25　ナンバー　1015

(74)代理人　弁理士　合田　潔　（外２名）

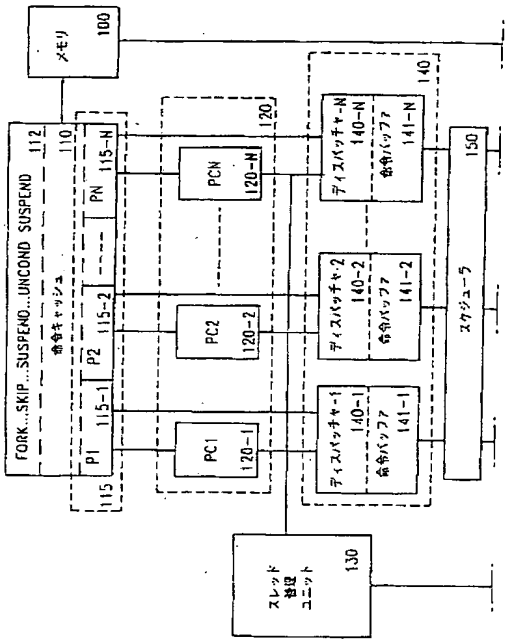
最終頁に続く

(54)【発明の名称】　推論並列命令スレッドの実行

(57)【要約】

【課題】　複数の命令スレッドの推論並列実行を可能にする、コンピュータ内の中央演算処理装置（ＣＰＵ）を提供する。

【解決手段】　このＣＰＵは、ＣＰＵの命令セットに追加され、並列実行用の潜在的な将来スレッドを示すために実行時以前にプログラムに挿入される、フォーク－中断命令を使用する。このＣＰＵは、１つまたは複数の命令キャッシュ・ポートを備えた命令キャッシュと、１つまたは複数のプログラム・カウンタからなるバンクと、１つまたは複数のディスパッチャからなるバンクと、スレッド間通信を処理し、依存関係に違反する将来スレッドを破棄するスレッド管理ユニットと、すべてのスレッドに共通する１組のアーキテクチャ化レジスタと、ＣＰＵ内の１つまたは複数の機能ユニット上での命令の並列実行をスケジューリングするスケジューラとを有する。



【特許請求の範囲】

【請求項１】コンピュータ内の中央演算処理装置において、

ａ．複数の命令を有する命令キャッシュ・メモリであって、その命令キャッシュが１つまたは複数の命令キャッシュ・ポートをさらに有する命令キャッシュ・メモリと、

ｂ．複数のプログラム・カウンタからなるプログラム・カウンタ・バンクであって、それぞれのプログラム・カウンタが命令キャッシュ内の１つまたは複数の命令を独立してアドレス指定し、アドレス指定した命令を命令キャッシュ・ポートの１つにポーティングすることができる、プログラム・カウンタ・バンクと、

ｃ．複数のディスパッチャからなり、それぞれのディスパッチャが命令バッファを１つずつ有し、それぞれのディスパッチャが１つまたは複数の命令キャッシュ・ポートから命令を受け取り、受け取った命令をその命令バッファに入れ、命令をデコードし、その関連バッファ内の命令間の依存関係を分析することができる、ディスパッチャ・バンクと、

ｄ．それぞれのスレッドがプログラム・カウンタの１つを使用して実行した一連の命令シーケンスを有する、１つまたは複数のスレッドをフォークし、０または１つ以上のスレッド間通信を処理するスレッド管理ユニットと、

ｅ．すべてのディスパッチャから命令を受け取り、１つまたは複数の機能ユニット上で実行するために命令をスケジューリングするスケジューラと、

ｆ．すべてのスレッド内の命令によってアクセス可能な１つまたは複数のアーキテクチャ化レジスタからなる固定セットを含むレジスタ・ファイルとを含み、

それにより、１つまたは複数の命令スレッドが機能ユニットによって並列に実行されることを特徴とする、中央演算処理装置。

【請求項２】プログラム・カウンタ・バンク内のプログラム・カウンタの１つがメイン・スレッド内の命令を追跡し、メイン・スレッドが順次追跡順序で最も早いスレッドであることを特徴とする、請求項１に記載の装置。

【請求項３】１つまたは複数のディスパッチャが、そのディスパッチャが分析中に解決できない１つまたは複数の依存関係について推測し、スレッド管理ユニットが、１つまたは複数の推測の結果として、プログラム依存関係の違反のために将来スレッドのいずれかによって実行された１つまたは複数の命令を破棄する必要があるかどうかを判定することができ、スレッド管理ユニットがこのような違反命令を破棄することを特徴とする、請求項２に記載の装置。

【請求項４】スレッド管理ユニットは、FORK命令が検出されたときに指定のアドレスから始まる将来スレッドをフォークすることができ、FORK命令がコンパイル時に命令スレッドに挿入され、FORK命令が１つまたは複数の将来スレッドの先頭を識別することを特徴とする、請求項３に記載の装置。

【請求項５】FORK命令が１つの命令コード・フィールドと１つまたは複数のアドレス・フィールドとを含み、それぞれのアドレスが将来スレッドの開始位置を識別することを特徴とする、請求項４に記載の装置。

【請求項６】FORK命令の命令コード・フィールドがビット０～５を含み、アドレス・フィールドがビット６～２９を含み、拡張命令コード・フィールドがビット３０と３１とを含むことを特徴とする、請求項５に記載の装置。

【請求項７】スレッド管理ユニットは、FORK命令が検出されたときに指定のアドレスから始まる将来スレッドをフォークすることができ、UNCOND_SUSPEND命令が検出されたときにその将来スレッドを無条件に中断し、FORK命令とUNCOND_SUSPEND命令がコンパイル時に挿入されることを特徴とする、請求項３に記載の装置。

【請求項８】FORK命令が１つの命令コード・フィールドと１つまたは複数のアドレス・フィールドとを含み、それぞれのアドレスが将来スレッドの開始位置を識別し、UNCOND_SUSPEND命令が１つの命令コード・フィールドを含むことを特徴とする、請求項７に記載の装置。

【請求項９】FORK命令の命令コード・フィールドがビット０～５を含み、アドレス・フィールドがビット６～２９を含み、拡張命令コード・フィールドがビット３０と３１とを含み、UNCOND_SUSPENDの命令コードがビット０～５を含む１つの１次命令コード・フィールドと、ビット２１～３１を含む１つの拡張命令コード・フィールドとを有することを特徴とする、請求項８に記載の装置。

【請求項１０】１つまたは複数のSUSPEND命令を有し、将来スレッドの１つの実行中にSUSPEND命令が検出され、SUSPEND命令に関連するコンパイル時指定条件が実行時に偽と評価された場合にスレッド管理ユニットがSUSPEND命令に関連する依存性領域内の１組の命令の結果を破棄し、SUSPEND命令がコンパイル時に挿入されることを特徴とする、請求項７に記載に装置。

【請求項１１】SUSPEND命令が、１つのSUSPEND命令コード・フィールドと、１つのモードビットと、１つの条件フィールドとを含むことを特徴とする、請求項１０に記載の装置。

【請求項１２】SUSPEND命令コードが、ビット０～５を含む１つの１次命令コード・フィールドと、ビット６を占有する１つのモード・フィールドと、ビット６～２０を占有し、それぞれが２ビットの長さの７つの条件サブフィールドから構成される１つの条件フィールドと、ビット２１～３１を含む１つの拡張命令コード・フィールドとを有することを特徴とする、請求項１１に記載の装置。

【請求項１３】スレッド管理ユニットは、FORK_SUSPEND

命令が検出されたときに指定のアドレスから始まる将来スレッドをフォークすることができ、FORK_SUSPEND命令がコンパイル時に命令スレッドに挿入され、FORK_SUSPEND命令が１組または複数組の命令を識別することができ、それぞれの組の命令がそれぞれの組の命令の有効実行を判定する関連条件を任意で有することを特徴とする、請求項３に記載の装置。

【請求項１４】FORK_SUSPEND命令が、１つの命令コード・フィールドと、１つのアドレス・フィールドと、１つまたは複数の条件フィールドとを含み、それぞれの条件フィールドが１つのカウント・フィールドと１つまたは複数の条件とを有することを特徴とする、請求項１３に記載の装置。

【請求項１５】FORK_SUSPEND命令が、ビット０～５を含む１つの命令コードと、ビット６～８を含む第１のカウント・フィールドと、第１のカウント・フィールドに関連し、ビット９～１０および１１～１２をそれぞれ含む２つの条件とを有する第１の条件フィールドと、ビット１３～１５を含む第２のカウント・フィールドと、第２のカウント・フィールドに関連し、ビット１６～１７および１８～１９をそれぞれ含む２つの条件とを有する第２の条件フィールドと、ビット２０～２９を含む１つのアドレス・フィールドと、ビット３０および３１を含む１つの拡張命令コード・フィールドとを有することを特徴とする、請求項１４に記載の装置。

【請求項１６】SKIP命令を検出したときに、将来スレッドが、SKIP命令によって指定された複数の命令をデコードし、実行を行わずに識別された命令の実行を引き受けることを特徴とする、請求項１０に記載の装置。

【請求項１７】SKIP命令が１つの命令コード・フィールドと１つのカウント・フィールドとを含むことを特徴とする、請求項１６に記載の装置。

【請求項１８】スレッド管理ユニットは、FORK_S_SUSPEND命令が検出されたときに指定のアドレスから始まる将来スレッドをフォークすることができ、FORK_S_SUSPEND命令がコンパイル時に命令スレッドに挿入され、FORK_S_SUSPEND命令が１組または複数組の命令を識別することができ、それぞれの組の命令がそれぞれの組の命令の有効実行を判定する関連条件を任意で有し、スレッドに開始時に複数の命令を識別するスキップ・カウント・フィールドをさらに有し、実行を行わずに識別された命令の実行を引き受けることを特徴とする、請求項３に記載の装置。

【請求項１９】FORK_S_SUSPEND命令が、１つの命令コード・フィールドと、１つのアドレス・フィールドと、１つのスキップ・カウント・フィールドと、１つまたは複数の条件フィールドとを含み、それぞれの条件フィールドが１つのカウント・フィールドと１つまたは複数の条件とを有することを特徴とする、請求項１８に記載の装置。

【請求項２０】スレッド管理ユニットは、FORK_M_SUSPEND命令が検出されたときに指定のアドレスから始まる将来スレッドをフォークすることができ、FORK_M_SUSPEND命令がコンパイル時に命令スレッドに挿入され、FORK_M_SUSPEND命令が１組のレジスタ・マスクを識別することができ、マスクに関連する条件がある場合にその条件が実行時に該当するのであれば、それぞれのマスクが有効ソース・オペランドを保持する複数のアーキテクチャ化レジスタからなるサブセットを識別することを特徴とする、請求項３に記載の装置。

【請求項２１】FORK_M_SUSPEND命令が、１つの命令コード・フィールドと、１つのアドレス・フィールドと、１つまたは複数の条件フィールドとを含み、それぞれの条件フィールドが１つのレジスタ・マスクと１つまたは複数の条件とを有することを特徴とする、請求項２０に記載の装置。

【請求項２２】FSKIP命令を検出したときに、将来スレッド・ディスパッチャが取出しと、その結果、この命令に続く指定の数の命令の実行をスキップし、FSKIP命令が有効オペランドを保持する１組のアーキテクチャ化レジスタを指定するレジスタ・マスクを識別することができ、メイン・スレッド・ディスパッチャがこれをＮＯＰとして扱い、FSKIP命令がコンパイル時に命令スレッドに挿入されることを特徴とする、請求項１０に記載の装置。

【請求項２３】FSKIP命令が、１つの命令コード・フィールドと、１つのマスク・フィールドと、１つのカウント・フィールドとを含むことを特徴とする、請求項２２に記載の装置。

【請求項２４】SKPMG命令を検出したときに、将来スレッドが、SKPMG命令によって指定された複数の命令をデコードし、実行を行わずに識別された命令の実行を引き受け、メイン・スレッド・ディスパッチャがこの命令を潜在的将来スレッドの開始アドレス用のマーカとして扱い、SKPMG命令がコンパイル時に命令スレッドに挿入されることを特徴とする、請求項１０に記載の装置。

【請求項２５】SKPMG命令が１つの命令コード・フィールドと１つのカウント・フィールドとを含むことを特徴とする、請求項２４に記載の装置。

【請求項２６】スレッド管理ユニットが任意でフォークすることができることを特徴とする、請求項１に記載の装置。

【請求項２７】命令キャッシュがメイン・メモリで置き換えられることを特徴とする、請求項１に記載の装置。

【請求項２８】中央演算処理装置を備えたコンピュータ・システム上で命令を実行する方法において、（ａ）複数の命令からなる静的シーケンスをコンパイル時に生成し、その命令の静的シーケンスを分析して１組のフォーク点を判定するステップと、（ｂ）コンパイル時に０個またはそれ以上のFORK命令を０個またはそれ以上のフォ

ーク点に挿入するステップと、（ｃ）メモリ内の固定位置から始まるメイン・メモリにその命令の静的シーケンスをロードし、その静的シーケンスのサブシーケンスを命令キャッシュに転送するステップと、（ｄ）現行アドレスから始まるメイン・プログラム・カウンタによりそのシーケンスをアドレス指定することにより、命令キャッシュから命令シーケンスをメイン・スレッドとして取り出し、現行アドレスから始まり、まだ組み合わされていない１つまたは複数の将来スレッドがあるかどうかを判定するために検査するステップと、（ｅ）組み合わされていない将来スレッドの妥当性を検査するステップと、（ｆ）０個またはそれ以上の組み合わされていない将来スレッドの有効に実行された部分をメイン・スレッドに組み合わせるステップと、（ｇ）取り出した命令をディスパッチャでデコードし、１つまたは複数の命令がFORK命令としてデコードされたかどうかを確認するために検査するステップと、（ｈ）命令がFORK命令以外の命令としてデコードされた場合に、命令依存関係を分析し、適切な機能ユニット上で実行するために命令をスケジューリングすることにより、メイン・スレッドを実行するステップと、（ｉ）完了ユニットにより命令実行を完了し、ステップ（ｄ）からこのステップまでを繰り返すステップと、（ｊ）命令がFORK命令としてデコードされた場合に、追加の将来スレッドをフォークするために使用可能なマシン資源があるかどうかを判定するために検査するステップと、（ｋ）使用可能資源がある場合に、FORK命令に関連するアドレスを将来プログラム・カウンタにロードすることにより、将来スレッドをフォークするステップと、（ｌ）それぞれメイン・プログラム・カウンタとメイン・スレッド・ディスパッチャの代わりに将来プログラム・カウンタの１つと将来スレッド・ディスパッチャの１つを使用することによってステップ（ｄ）～（ｈ）を実行することにより、フォーキング・スレッドの実行と並列して将来スレッドを実行し、将来スレッドがメイン・スレッドと組み合わされるか、または将来スレッドがスレッド管理ユニットによって消去された場合に、将来スレッドの実行を中断するステップとを含むことを特徴とする方法。

【請求項２９】前記ステップ（ｂ）が、

（ｂ．１）　すべての将来スレッドの終わりにUNCOND_SUSPEND命令を挿入するという追加のサブステップを有し、

前記ステップ（ｌ）が、

（ｌ．１）　UNCOND_SUSPEND命令を検出したときに将来スレッドの実行を中断するという追加のサブステップを有し、

前記ステップ（ｈ）が、

（ｈ．１）　その対応将来スレッド以外のスレッドによって実行するために検出された場合に、UNCOND_SUSPEND命令をＮＯＰとして扱うという追加のサブステップを有することを特徴とする、

請求項２８に記載の方法。

【請求項３０】前記ステップ（ｂ）が、

（ｂ．２）　すべてのUNCOND_SUSPEND命令に対応する０個またはそれ以上のSUSPEND命令を挿入するという追加のサブステップを有し、

前記ステップ（ｌ）が、

（ｌ．２）　SUSPEND命令に関連するコンパイル時指定条件が実行時に偽と評価された場合に、SUSPEND命令に関連する依存性領域内の１組の命令を破棄するという追加のサブステップを有し、

前記ステップ（ｈ）が、

（ｈ．２）　その対応将来スレッド以外のスレッドによって実行するために検出された場合に、SUSPEND命令をＮＯＰとして扱うという追加のサブステップを有することを特徴とする、

請求項２９に記載の方法。

【請求項３１】前記ステップ（ｂ）が、

（ｂ．３）　０個またはそれ以上のSKIP命令を将来スレッドに挿入するという追加のサブステップを有し、

前記ステップ（ｌ）が、

（ｌ．３）　SKIP命令に続く指定の数の命令をデコードし、将来スレッドとしての実行中に、実行を行わずにその指定の数の命令の実行を引き受けるという追加のサブステップを有し、

前記ステップ（ｈ）が、

（ｈ．３）　その対応将来スレッド以外のスレッドによって実行するために検出された場合に、SKIP命令をＮＯＰとして扱うという追加のサブステップを有することを特徴とする、

請求項３０に記載の方法。

【請求項３２】前記ステップ（ｂ）が、

（ｂ．４）　０個またはそれ以上のFSKIP命令を将来スレッドに挿入するという追加のサブステップを有し、

前記ステップ（ｌ）が、

（ｌ．４）　将来スレッドとしての実行中にFSKIP命令に続く指定の数の命令の取出しをスキップし、関連マスクで識別されたレジスタに有効オペランドを保持するものとしてマークを付けるという追加のサブステップを有し、

前記ステップ（ｈ）が、

（ｈ．４）　その対応将来スレッド以外のスレッドによって実行するために検出された場合に、FSKIP命令をＮＯＰとして扱うという追加のサブステップを有することを特徴とする、

請求項３０に記載の方法。

【請求項３３】前記ステップ（ｂ）が、

（ｂ．５）　すべての将来スレッドの先頭にSKPMG命令を挿入するという追加のサブステップを有し、

前記ステップ（ｌ）が、

（ｌ．５）　SKIP命令に続く指定の数の命令をデコードし、将来スレッドとしての実行中に、実行を行わずにその指定の数の命令の実行を引き受けるという追加のサブステップを有し、
前記ステップ（ｈ）が、
（ｈ．５）　その対応将来スレッド以外のスレッドによって実行するためにSKPMGが検出された場合に、SKPMG命令の命令アドレス以降について、過去に将来スレッドがフォークされたかどうかを判定するために検査するという追加のサブステップを有し、
前記ステップ（ｄ）が、
（ｄ．１）　メイン・プログラム・カウンタによりそのシーケンスをアドレス指定することにより、命令キャッシュから命令シーケンスを取り出すというステップで置き換えられることを特徴とする、
請求項３０に記載の方法。
【請求項３４】前記ステップ（ｂ）が、
（ｂ．６）　０個またはそれ以上のFORK_SUSPEND命令を０個またはそれ以上の潜在的フォーク点に挿入するというステップで置き換えられ、
前記ステップ（ｊ）が、
（ｊ．１）　命令がFORK_SUSPEND命令としてデコードされた場合に、追加の将来スレッドをフォークするために使用可能なマシン資源があるかどうかを判定するために検査するというステップで置き換えられ、
前記ステップ（ｋ）が、
（ｋ．１）　使用可能資源がある場合に、FORK_SUSPEND命令に関連するアドレス（複数も可）を将来プログラム・カウンタ（複数も可）にロードすることにより、将来スレッドをフォークするというステップで置き換えられ、
前記ステップ（ｌ）が、
（ｌ．６）　関連するコンパイル時指定条件が実行時に真に該当しない場合に、将来スレッド内の命令の一部または全部の結果を破棄するという追加のサブステップを有することを特徴とする、
請求項２８に記載の方法。
【請求項３５】前記ステップ（ｂ）が、
（ｂ．７）　０個またはそれ以上のFORK_S_SUSPEND命令を０個またはそれ以上の潜在的フォーク点に挿入するというステップで置き換えられ、
前記ステップ（ｊ）が、
（ｊ．２）　命令がFORK_S_SUSPEND命令としてデコードされた場合に、追加の将来スレッドをフォークするために使用可能なマシン資源があるかどうかを判定するために検査するというステップで置き換えられ、
前記ステップ（ｋ）が、
（ｋ．２）　使用可能資源がある場合に、FORK_S_SUSPEND命令に関連するアドレス（複数も可）を将来プログラム・カウンタ（複数も可）にロードすることにより、将来スレッドをフォークするというステップで置き換えられ、
前記ステップ（ｌ）が、
（ｌ．７）　将来スレッドの先頭にある指定の数の命令をデコードし、これらの命令の実行を行わずに指定の数の命令の実行を引き受け、関連するコンパイル時指定条件が実行時に真に該当しない場合に、将来スレッド内の命令の一部または全部の結果を破棄するという追加のサブステップを有することを特徴とする、
請求項２８に記載の方法。
【請求項３６】前記ステップ（ｂ）が、
（ｂ．８）　０個またはそれ以上のFORK_M_SUSPEND命令を０個またはそれ以上の潜在的フォーク点に挿入するというステップで置き換えられ、
前記ステップ（ｊ）が、
（ｊ．３）　命令がFORK_M_SUSPEND命令としてデコードされた場合に、追加の将来スレッドをフォークするために使用可能なマシン資源があるかどうかを判定するために検査するというステップで置き換えられ、
前記ステップ（ｋ）が、
（ｋ．３）　使用可能資源がある場合に、FORK_M_SUSPEND命令に関連するアドレス（複数も可）を将来プログラム・カウンタ（複数も可）にロードすることにより、将来スレッドをフォークするというステップで置き換えられ、
前記ステップ（ｌ）が、
（ｌ．８）　命令のソース・レジスタ・オペランドに関連するコンパイル時指定条件が実行時に真に該当しない場合に、将来スレッド内の命令の一部または全部の結果を破棄するという追加のサブステップを有することを特徴とする、
請求項２８に記載の方法。
【請求項３７】前記ステップ（ｈ）が、
（ｈ．６）　スレッド実行中のすべての分岐解決をＴＭユニットに連絡し、ＴＭユニットがこの情報を使用して、間違った分岐アドレスにフォークされた将来スレッドとすべての依存スレッドを破棄する必要があるかどうかを判定するという追加のサブステップを有することを特徴とする、
請求項２８に記載の方法。
【請求項３８】前記ステップ（ｄ）が、
（ｄ．２）　ＴＭユニットが、事前フォーク済みスレッドのいずれかが既定のタイムアウト期間より長い間、組み合わされない状態を維持していたかどうかを判定するために検査し、このようなスレッドを破棄するという追加のサブステップを有することを特徴とする、
請求項２８に記載の方法。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、コンピュータ・シ

ステム上での複数の命令からなる並列スレッドの実行の分野に関する。より具体的には、本発明は、コンピュータ・プログラム内のどの命令スレッドを並列に実行できるかを判定し、この並列実行を推論方式で実施することに関する。

【０００２】

【従来の技術】市販のマイクロプロセッサは、現在、単一プロセッサ・アーキテクチャを使用している。このアーキテクチャは、共通する１組のアーキテクチャ上可視のレジスタを共用する１つまたは複数の機能ユニット（分岐ユニット、ロード／ストア・ユニット、整数演算ユニット、浮動小数点演算ユニットなど）を含むことができる。（レジスタは、そのプロセッサのアセンブリ・レベルのプログラマか、またはより上位レベルのプログラムをマシンのアセンブリ・レベルに変換するプロセッサのコンパイラにとってアクセス可能であれば、アーキテクチャ上可視であると見なされる。）

【０００３】コンピュータ・システムでは、コンパイラまたはアセンブリ・プログラマにより生成された命令は、実行時以前に命令メモリに１つのシーケンスとして置かれ、そこから取り出して実行することができる。このシーケンスは静的順序と呼ばれている。動的順序は、コンピュータがこのような命令を実行する順序である。動的順序は静的順序である場合もあれば、そうではない場合もある。（以下の説明で使用する「コンパイル時」という表現は、実行時以前の処理のタイミングを意味する。ただし、このような処理はコンパイラによって行われる可能性が非常に高いが、アセンブリ・レベル・プログラミングなどの他の手段が代わりに使用されることもあることに留意されたい。）

【０００４】先行技術のスカラー・コンピュータすなわち非スーパースカラー・コンピュータ、または一度に１つずつ命令を実行するマシンには、順次追跡順序と呼ばれる固有の動的実行順序が備わっている。この順次追跡順序で命令Ａを別の命令Ｂより先行させると、このような命令Ａは、命令Ｂより早期の命令とも呼ばれる。このようなコンピュータは、制御命令が検出されるまで、その静的順序で命令を実行する。この検出時点では、元の順次順序から外れた（不連続）位置から命令を取り出すことができる。その後、次の制御命令が検出されるまで、命令はもう一度、静的順次順序で実行される。制御命令とは、以降の命令取出しを不連続位置から強制的に始めさせることにより、順次命令取出しを変更する可能性のある命令である。制御命令としては、分岐、ジャンプなどの命令がある。

【０００５】先行技術のマシンの中には、プログラムの依存関係に違反しなければ、順次追跡順序から外れた命令を実行できるものもある。このようなマシンは、順次追跡順序の順に命令を取り出すか、または順次追跡順序で同時に命令グループを取り出す。しかし、このようなマシンは、順次追跡順序から外れた命令を取り出すことはしない。たとえば、順次追跡順序で命令Ａが命令Ｂより先行する場合、先行技術のマシンでは、命令Ａの次に命令Ｂを順に取り出すか、または命令ＡとＢを同時に取り出すことができるが、命令Ａより前に命令Ｂを取り出すことはない。このような制約は単一プログラム・カウンタを備えたマシンの特徴である。したがって、このような制約のあるマシンは、単一スレッドまたはユニスレッド・マシンと言われている。このようなマシンは、順次追跡順序で先行する命令を取り出す前に順次追跡順序で後続の命令を取り出すことができない。

【０００６】発明者らが把握している現行世代の市販マイクロプロセッサは、単一スレッドの制御の流れを備えている。このようなプロセッサは、所与のプログラムの様々な部分の制御とデータからの独立性を活用する能力が制限されている。重要な制限としては、以下のものがある。

○　単一スレッドとは、マシンが複数の命令からなる単一シーケンスの取出しに制限され、プログラム制御の複数の流れ（スレッド）を同時に追跡できないことを意味する。

○　さらに単一スレッド制御とは、データ非依存命令がまとめて取り出すべきスレッド内で時間的に十分接近していて（たとえば、複数の命令を命令バッファに同時に取り出す場合）、データ独立性を検出するためにまとめて検査される場合のみ、データ独立性を活用できることを意味する。

○　上記の制限は、制御非依存命令とデータ非依存命令をまとめてグループ化するためにコンパイラを頼りにすることを意味する。

○　先行技術のマイクロプロセッサの中には、制御流れ推測という何らかの形態の制御命令（分岐）予測を含むものがある。この場合、制御命令の結果が正しく推測されたと期待して、順次追跡順序で制御命令に続く命令を取り出して実行することができる。制御流れの推測は、より高度の並列性を活用するために必要な技法としてすでに認められている。しかし、制御への依存性に関する知識が欠けているため、単一スレッド動的推測は、制御流れ推測誤り（不良推測）が発生するまで先読みする能力しか拡張することができない。不良推測が行われると、多くの実行サイクルが浪費される可能性がある。制御依存性分析のハードウェア・コストを無視しても、単一スレッド制御流れ推測により制御への依存性を実行時に学習することはよく見ても適用範囲が制限されることに留意されたい。ここで使用する適用範囲とは、命令間の制御およびデータの依存関係を同時に検査することができる命令の数を意味する。通常、実行時よりコンパイル時に得られる適用範囲の方がかなり大きくなる可能性がある。

○　制御流れに関するコンパイル時推測は、実行時推測

より適用範囲がかなり大きくなる可能性があるが、制御依存性分析から利益を得る場合もある。しかし、単一スレッドの実行時制限により、コンパイラは、実行時に並列性が活用可能になるように非推論命令とともに推論命令をまとめてグループ化する必要がある。

【０００７】実行時により多くの並列性を発揮するためにコンパイル時制御流れ推測を使用することについては、上記に述べた。現行マシンのコンパイラは、この推測をコード化する能力が制限されている。保護と格上げのような一般に使用される手法では、単一スレッド実行の早期に推論実行すべき一部の命令をパーコレーションするためにコンパイラを頼りにしている。また、このような手法では、制御流れ推測を推論命令にコード化する必要もある。この手法には、次のような重要な制限がある。

○　通常、かなり浅い制御流れ推測をコード化するためにすべての命令で十分な未使用ビットを見つけることは非常に難しいことである。逆方向互換性の制約（いかなる変換も行わずに古い２進コードを実行できる能力）があるため、制御流れ推測のコード化を含めるために命令コード化を任意に再配置（新しいアーキテクチャを意味する）することができないことに留意されたい。

○　前述のパーコレーション技法は、推測誤りを処理するために余分なコードまたはコード・コピーあるいはその両方を必要とする場合が多い。その結果、コードが拡大する。

○　アーキテクチャ上、推論命令によって発生する例外の順次処理や、割込みの精密処理が必要になる場合が多い。しかし、前述のパーコレーション技法は上方推論コード・モーションを使用するので、このような順序外推論実行のコンテキストでこれらを実施することは非常に難しい場合が多い。パーコレーションした命令を区別し、その元の位置を追跡するには、特殊な機構が必要である。ただし、外部命令の観点から見ると、精密割込み処理の制約下では順次追跡順序から外れた命令実行は推論的と見なすことができることに留意されたい。しかし、制約はあるもののより広く使用される意味では、命令（より正確には、その命令の特定の動的インスタンス）が順次追跡順序の一部であると設定する前に命令処理が開始される場合、またはある命令のオペランドの妥当性を確立する前にそのオペランドが提供される場合に、実行が推論的と見なされる。

【０００８】制御依存性を知らないと、ネストされたループのパフォーマンスにとって特に高価なものになる場合がある。たとえば、外側反復がデータ依存の内側ループ反復とは無関係の制御とデータであるようなネストされたループについて検討する。外側ループ反復とは無関係の制御およびデータの知識が活用されない場合は、内側ループに関する順次制御流れ推測のため、その取出しおよび実行を遅延させなければならない。さらに、このように制御依存性の知識が欠けているため、制御およびデータ非依存の内側ループ反復の１つの予測を誤ったときに、外側ループから推論実行される命令を不必要に破棄する場合もある。また、内側ループの制御流れがデータに依存し、そのため、極めて予測不能である場合には、内側ループの制御流れ推測について予測誤りが発生する確率が非常に高くなる可能性があることにも留意されたい。このような例としては、以下のものがある。

【０００９】/* 環境リストを検査する */

```
for (fp = xlenv; fp; fp = cdr (fp) )
for (ep = car (fp); ep; ep = cdr (ep) )
if (sym == car (car (ep) ) )
cdr (car (ep) ) = new_p;
```

【００１０】これは二重にネストされたループであって、内側ループは連係リストを詳しく検討し、その反復はどちらも前の反復に依存する制御とデータである。しかし、内側ループのそれぞれの活動化（すなわち、外側ループの反復）は前の反復とは無関係である。［これは、SPECint92ベンチマークの１つ（Li）で最も頻繁に実行されるループの１つ（Xlgetvalue）をわずかに変更したバージョンである。］

【００１１】前述のように、単一制御流れを備えたマシンは、推論または非推論あるいはその両方のデータ非依存命令をまとめてグループ化するためにコンパイラに頼らなければならない。しかし、すべてのデータおよび制御非依存命令を効率よくまとめてグループ化するには、コンパイラは、適切なコード化のためにアーキテクチャ化した十分なレジスタを必要とする。このため、レジスタ圧力は高まり、追加のスピル・コードのオーバヘッドのためにこのようなコード・モーションが実を結ばなくなるような点を上回っている。

【００１２】大規模並列アーキテクチャの実現を主な目的とする、複数スレッドを備えたプロセッサを構築するために、研究の試みがいくつか行われてきた。複数スレッドを管理するというオーバヘッドは、実行の並行性が追加されるというパフォーマンス上の利得を潜在的に上回る可能性がある。スレッド管理に関連するオーバヘッドとしては、以下のものがある。

○　明示または暗黙同期プリミティブによる、データおよび制御依存性のための部分順序の管理と通信

○　別のスレッドが使用するためにあるスレッドが生成した値の通信

○　静的すなわちコンパイル時のスレッド・スケジューリングと、動的すなわち実行時のスレッド・スケジューリングとの対照に関連するトレードオフ。静的スレッド・スケジューリングは、実行時のハードウェアを簡略化するが、フレキシビリティが低く、実現したマシンのスレッド資源をコンパイラに公表してしまうため、各種の実施態様に対応する再コンパイルを必要とする。これに対して、動的スレッド・スケジューリングは様々な実施

態様に適応可能で、いずれも同じ実行可能命令を共用するが、追加の実行時ハードウェア・サポートを必要とする。

【００１３】定義

さらに詳述する前に、以下の一連の作業定義は非常に有用である。

○　スレッド：　単一命令順序付け制御（単一プログラム・カウンタを意味する）と共用される１組のアーキテクチャ上可視のマシン状態とを使用して実行可能な命令のシーケンス。

○　順次追跡順序：プログラム命令の実行シーケンスの動的順序であって、一度に１つずつ命令を実行する単一制御スレッド非推論マシン上でそのプログラムを完全実行した結果得られるもの。

○　メイン・スレッドと将来スレッド：所与の時点での１組のスレッドのうち、順次追跡順序で最も早い命令を実行するスレッドをメイン・スレッドという。残りのスレッドは将来スレッドという。

【００１４】

【発明が解決しようとする課題】本発明の一目的は、様々な命令スレッドを同時に取り出して実行するための改良された方法および装置である。

【００１５】本発明の一目的は、１つまたは複数の制御およびデータ依存関係を備えた様々な命令スレッドを同時に取り出して実行するための改良された方法および装置である。

【００１６】本発明の一目的は、１つまたは複数の制御およびデータ依存関係を備えた様々な命令スレッドを同時に取り出して推論実行するための改良された方法および装置である。

【００１７】本発明の一目的は、コンピュータ・アーキテクチャの様々な実施態様上で１つまたは複数の制御およびデータ依存関係を備えた様々な命令スレッドを同時に取り出して推論実行するための改良された方法および装置である。

【００１８】

【課題を解決するための手段】本発明は、複数の命令スレッドの推論並列実行を可能にする、コンピュータ内の中央演算処理装置（ＣＰＵ）の改良策である。本発明では、ＣＰＵの命令セットに追加され、並列実行のために潜在的将来スレッドを示すために実行時以前にプログラムに挿入される、新規のフォーク－中断命令を開示する。これは、コンパイラによって実行されることが好ましい。

【００１９】ＣＰＵは、１つまたは複数の命令キャッシュ・ポートを備えた命令キャッシュと、その命令キャッシュ内の命令を独立してアドレス指定できる１つまたは複数のプログラム・カウンタからなるバンクとを有する。プログラム・カウンタが１つの命令をアドレス指定すると、アドレス指定された命令は、命令キャッシュ・ポートにポートされる。さらにＣＰＵは、１つまたは複数のディスパッチャも有する。ディスパッチャは、命令キャッシュ・ポートにポートされた命令をそのディスパッチャに関連する命令バッファで受け取る。また、ディスパッチャは、そのバッファ内の命令間の依存関係も分析する。ＣＰＵ内のスレッド管理ユニットは、すべてのスレッド間通信を管理し、プログラム依存関係に違反する将来スレッドを破棄する。ＣＰＵスケジューラは、ＣＰＵ内のすべてのディスパッチャから命令を受け取り、ＣＰＵ内の１つまたは複数の機能ユニットでの命令の並列実行をスケジューリングする。通常、メイン・プログラム・スレッド内の命令の実行は１つのプログラム・カウンタが追跡し、将来スレッドの並列実行は残りのプログラム・カウンタが追跡することになる。命令のポーティングとその機能ユニット上での実行は、推論方式で実行することができる。

【００２０】

【発明の実施の形態】本発明は、複数のプログラム位置からの命令を同時に取り出して、推論し、実行し、その結果、複数の制御スレッドを追跡するように従来の単一スレッド推論スーパースカラーＣＰＵを強化するために、フォーク－中断命令を提案している。

【００２１】図１は、本発明で提案した方法を実行すると思われる典型的なプロセッサ編成のハードウェアのブロック図である。実行方法については、後述する。図１の詳細説明は以下の通りである。

【００２２】ブロック１００は、プロセッサ上で実行するためのプログラム・データおよび命令を保持する、プロセッサの中央演算処理装置（ＣＰＵ）のメモリ・ユニットである。このメモリ・ユニットは、このメモリ・ユニットの頻繁に使用する命令およびデータ部分が通常、命令キャッシュ・ユニット（ブロック１１０）とデータ・キャッシュ・ユニット（ブロック１７０）にそれぞれ保管されるように、キャッシュ・ユニットとのインタフェースが取られている。あるいは、命令キャッシュとデータ・キャッシュは、単一一体化ユニットに統合することができる。キャッシュ・ユニットのアクセス時間は、通常、メモリ・ユニットのアクセス時間よりかなり短い。上記のようなメモリ・ユニットとキャッシュ・ユニットは当技術分野では周知のものである。たとえば、メイン・メモリとそのポートをキャッシュ・メモリとそのポートに使用することにより、キャッシュ・ユニットを置き換えることができる。また、キャッシュは、周知のように、複数のキャッシュまたは１つまたは複数のレベルを備えたキャッシュで構成することもできる。

【００２３】ブロック１１０は、プロセッサ上で実行するためのプログラム命令を保持する、プロセッサ（ＣＰＵ）の命令キャッシュ・ユニットである。これは、FORK、SKIP、SUSPEND、UNCOND_SUSPEND（ブロック１１２）など、本発明で提案した新しい命令を含む。上記および

その他の新しい命令の意味の詳細については、後述する。

【００２４】命令キャッシュの複数ポートＰ１、Ｐ２、・・・、ＰＮ（ブロック１１５－１、１１５－２、・・・、１１５－Ｎ）を含むブロック１１５は、現行技術では新規のものである。この複数ポートにより、並列実行される命令スレッドへの命令の同時ポーティングが可能になる。あるいは、単一幅広ポートを使用して所与のスレッドに複数の命令をポートし、そのスレッドがポートされた命令を実行していて使用中の間に、同じポートを使用して、複数の命令を別のスレッドにポートすることも可能である。

【００２５】ブロック１２０は、プログラム・カウンタＰＣ１、ＰＣ２、・・・、ＰＣＮ（ブロック１２０－１、１２０－２、・・・、１２０－Ｎ）からなるバンクである。これらのカウンタは、当技術分野で周知であれば、どのカウンタであってもよい。それぞれのプログラム・カウンタは所与のスレッドの実行を追跡する。現在までに設計されたすべての商用ＣＰＵは、所与のプログラムについて、単一命令スレッドの実行だけを制御する必要がある。このため、現行技術および従来技術は単一プログラム・カウンタに制限されており、したがって、複数プログラム・カウンタからなるバンクは本発明の新規の態様である。それぞれのプログラム・カウンタは、命令キャッシュ内の１つまたは複数の連続命令をアドレス指定することができる。図１および図２のブロック図に示す好ましい実施例では、それぞれのプログラム・カウンタが１つの命令キャッシュ・ポートに関連づけられている。あるいは、各種のプログラム・カウンタが１つの命令キャッシュ・ポートを共用することもできる。

【００２６】さらに、好ましい実施例では、特定のプログラム・カウンタがメイン・スレッドに関連づけられ、残りのプログラム・カウンタが将来スレッドの実行を追跡する。図１のＰＣ１（ブロック１２０－１）は、メイン・スレッド・プログラム・カウンタである。残りのプログラム・カウンタは、将来スレッド・プログラム・カウンタ（ブロック１２０－２、・・・、１２０－Ｎ）と呼ぶ。

【００２７】ブロック１３０は新規のスレッド管理（ＴＭ）ユニットを示しているが、これは、新しいスレッドをフォークすることができる新しい命令の実行と、組合せプロセス（後述する）によるスレッド間通信の処理とを担当する。

【００２８】また、このユニットは、１つまたは複数の将来スレッドの命令の一部または全部を破棄することもできる。さらにこのユニットは、１つまたは複数の推測の結果として、いずれかの将来スレッドが実行した１つまたは複数の命令をプログラム依存関係の違反のために破棄する必要があるかどうかを判定することもできる。実行時に推測が行われる場合は、推測ユニットによってＴＭユニットにそれが連絡される。たとえば、ディスパッチャ・ブロック（後述するブロック１４０）内の分岐命令結果の推測は、ＴＭユニットに連絡する必要がある。コンパイル時に推測が行われ、命令としてコード化される場合は、このような命令をデコードするブロック１４０のディスパッチャによってそれがＴＭユニットに連絡される。その結果、推論方式で複数スレッドを実行できるようになることは、本発明の固有の特徴である。

【００２９】また、メイン・スレッドと将来スレッドの並列取出しおよび実行とは、提案したマシンがその順次追跡順序から外れる命令を取り出して実行できることを意味することにも留意されたい。このマシンのこのような固有の特徴により、単一プログラム・カウンタのためにその順次追跡順序から外れる命令を取り出すことができない先行技術のマシンとは区別される。

【００３０】ブロック１４０は、ディスパッチャー１、ディスパッチャー２、・・・、ディスパッチャーＮ（ブロック１４０－１、１４０－２、・・・、１４０－Ｎ）という複数のディスパッチャからなるバンクを示し、それぞれのディスパッチャは１つの特定のプログラム・カウンタに関連づけられ、その結果、そのディスパッチャに関連する命令バッファ（ブロック１４１－１、１４１－２、・・・、１４１－Ｎ）で命令キャッシュ・ポートの１つから命令を受け取ることができる。また、ディスパッチャは、そのバッファ内の命令間の依存関係をデコードして分析することもできる。さらにディスパッチャは、後述するSKIP、FSKIP、またはSKPMGの各命令の意味の実現も担当する。

【００３１】ディスパッチャが検出する命令は、スレッドをフォークまたは中断することができるが、スレッド管理ユニット（ブロック１３０）に転送される。このＴＭユニットは、対応するプログラム・カウンタに適切な開始命令をロードすることにより、将来スレッドの活動化を担当する。また、ＴＭユニットは、UNCOND_SUSPEND命令を検出したときに将来スレッド・ディスパッチャを中断する。

【００３２】順序外実行のための実行時依存性分析の実施技法は、当技術分野では周知である。メイン・プログラム・カウンタに関連し、そのため、メイン・スレッドに関連するディスパッチャは、メイン・スレッド・ディスパッチャと呼ばれる。図１のディスパッチャー１（ブロック１４０－１）はメイン・スレッド・ディスパッチャである。残りのディスパッチャ（ブロック１４０－２、・・・、１４０－Ｎ）は将来プログラム・カウンタおよび将来スレッドに関連し、将来スレッド・ディスパッチャと呼ばれる。

【００３３】本発明で提案するディスパッチャのバンクの新規の態様は、１つのディスパッチャのバッファ内の命令の実行時依存性分析を他のディスパッチャのものとは無関係に（したがって並列に）実行できる点である。

これは、指定の条件下で命令スレッドの非依存性を保証することができるコンパイル時依存性分析によって可能になる。したがって、一方で、実行時依存性分析はコンパイル時分析の方が潜在的に適用範囲がかなり広いことによる恩恵を受けることになる（適用範囲が広いということは、相互依存性について多数の命令を同時に検査できる能力を意味する）。もう一方で、コンパイル時分析はフォーク－中断機構による恩恵を受けるが、この機構により実行時結果に関する推測によって独立スレッドを明示的に識別することができる。実行時またはコンパイル時の依存性分析技法は当技術分野では周知であるが、実行時依存性分析ハードウェアにコンパイル時依存性分析を明示推論方式で連絡することは、本発明の新規な点である。

【００３４】ブロック１５０は、ディスパッチャのバンク（ブロック１４０）内のすべてのディスパッチャから命令を受け取って、機能ユニット（ブロック１８０）の１つで実行するためにそれぞれの命令をスケジューリングする、スケジューラである。１つまたは複数のディスパッチャから同一サイクル中に受け取ったすべての命令は、互いに独立していると想定する。このようなスケジューラも、スーパースカラー・マシンの先行技術では周知である。代替実施例では、このスケジューラを１組のスケジューラに分割し、それぞれが機能ユニット（ブロック１８０）の規定のサブセットを制御するようにすることもできる。

【００３５】ブロック１６０は、１組のレジスタ・セットを含むレジスタ・ファイルである。このセットは、アーキテクチャ上可視のレジスタ・セットと、アーキテクチャ上不可視のレジスタとにさらに分割される。アーキテクチャ上可視すなわちアーキテクチャ化したレジスタとは、マシンのアセンブリ・レベルのプログラマ（またはコンパイラ）にとってアクセス可能なレジスタの固定セットを意味する。レジスタ・ファイルのアーキテクチャ上可視のサブセットは、通常、すべてのスレッド（メイン・スレッドと将来スレッド）に共通するはずである。アーキテクチャ上不可視のレジスタはＣＰＵの様々な物理レジスタを含み、そのサブセットはアーキテクチャ化したレジスタにマッピングされる。すなわち、アーキテクチャ化したレジスタに関連する値を含む。レジスタ・ファイルは、多くの命令を実行するための機能ユニットにオペランドを提供し、実行の結果も受け取る。このようなレジスタ・ファイルは先行技術では周知である。

【００３６】組合せプロセス（後述する）の実施態様の一部として、ＴＭユニット（ブロック１３０）はレジスタ・ファイルとやりとりし、組合せ後にすべてのアーキテクチャ化したレジスタが適切な非アーキテクチャ化物理レジスタに関連づけられるようにする。

【００３７】ブロック１７０は、命令によってソース・オペランドとして使用されるデータ値の一部と、実行された命令によって生成されるデータ値の一部とを保持する、プロセッサのデータ・キャッシュ・ユニットである。複数の機能ユニットが複数のメモリ常駐データ値を同時に要求し、複数のメモリ束縛結果を同時に生成する可能性があるので、データ・キャッシュは通常、マルチポート化されているはずである。マルチポート化データ・キャッシュは先行技術では周知である。

【００３８】ブロック１８０は、複数の機能ユニット（機能ユニット－１、機能ユニット－２、機能ユニット－Ｋ）からなるバンクであり、それぞれのユニットは一部またはすべてのタイプの命令を実行することができる。機能ユニットは、レジスタ・ファイル（ブロック１６０）またはデータ・キャッシュ（ブロック１７０）から入力ソース・オペランドを受け取り、そこに出力結果を書き込む。図１および図２に示す好ましい実施例では、すべての機能ユニットが同一であり、そのため、どのような命令も実行することができる。あるいは、バンク内の複数の機能ユニットを非対称にして、特定のユニットが所与のサブセットの命令だけを実行できるようにすることもできる。スケジューラ（ブロック１５０）は、この非対称を認識し、適切に命令をスケジューリングする必要がある。このようなトレードオフも先行技術では一般的である。

【００３９】ブロック１９０は、アーキテクチャによって有効な順序と見なされる順序での命令実行の完了を担当する命令完了ユニットである。ＣＰＵが順序外の命令を実行できるとしても、それを同一順序で完了できるかどうかは、アーキテクチャ上の制約による。将来スレッド・ディスパッチャによる実行のためにスケジューリングされた命令は、推論スレッドの場合にＴＭユニット（ブロック１３０）が将来スレッドの妥当性を確認したあとでのみ、完了ユニットによる完了の対象となる。

【００４０】本発明では、コンパイル時に命令シーケンスに挿入可能な新しい命令をいくつか提案する。このような命令の意味の詳細は以下の通りである。

【００４１】１．FORK

この命令は、１つまたは複数の命令スレッドの開始アドレス（複数も可）を識別する。識別されたそれぞれの命令スレッドは将来スレッドと呼ばれる。このような将来スレッドは、FORKのあとに順に続く一連の命令シーケンスを引き続き実行するフォーキング・スレッドと同時に実行することができる。将来スレッド用の開始ＣＰＵ状態は、FORK命令を検出した時点のＣＰＵ状態のコピーである。

【００４２】２．UNCOND_SUSPEND

この命令を検出すると、将来スレッドは、無条件にそのスレッドを中断し、フォーキング・スレッドとの組合せを待たなければならない。これは、たとえば、無条件中断命令後の命令が別のスレッド上の何らかの命令との重

要なデータ依存性を有する場合に必要になる可能性がある。提案したこの命令は他のいかなる属性も必要としないので、SUSPEND命令（後述する）とも組み合わせることが可能である。すなわち、SUSPEND命令のコード化の１つは単に無条件中断を指定することができるだけになる。

【００４３】３．SUSPEND

この命令を検出すると、将来スレッドは、引き続きその命令取出しと実行を続行することができるが、第１のSUSPEND命令に関連するコンパイル時指定条件が実行時に偽と評価された場合は、そのプログラムの順次追跡順序での第１のSUSPEND命令と第２のSUSPEND命令またはUNCOND_SUSPEND命令との間の一連の命令シーケンスの結果が破棄される。

【００４４】以下の命令を簡略化するため、SUSPEND命令の依存領域という用語は、SUSPEND命令後の最初の命令から始まり、他のSUSPEND命令を検出した時点またはUNCOND_SUSPEND命令を検出時点に終了する、順次追跡順序での一連の命令シーケンスであると定義する。

【００４５】４．SKIP

この命令を検出すると、将来スレッドは、次のコンパイル時指定数の命令（通常はスピル・ロード）をデコードし、対応するソース・レジスタと宛先レジスタに有効のマークを付けることによってこれらの命令の実行を引き受けることができるが、このスレッドはその命令に関連する動作を実際に実行する必要はない。メイン・スレッドはこの命令をＮＯＰとして取り扱う。

【００４６】５．FORK_SUSPEND

この命令の命令コードは、将来スレッドの先頭を識別するアドレスと、一連の数値（Ｎ１、Ｎ２、・・・、Ｎｎ）とに関連づけられ、それぞれの数値には条件が付いている場合もあれば付いていない場合もある。所与の一連のｎ個の数値は、FORK命令に関連するアドレスから始まる命令のｎ個の連続グループを意味する。関連条件が一切付いていない数値は、対応するグループの命令を将来スレッドとして無条件に実行できることを意味する。関連条件が付いている数値は、コンパイル時指定条件が実行時に真と評価された場合のみ、対応するグループの命令の将来スレッドによる実行が有効になるはずであることを意味する。

【００４７】６．FORK_S_SUSPEND

この命令の命令コードは、将来スレッドの先頭を識別するアドレスと、数値ｓと、一連の数値（Ｎ１、Ｎ２、・・・、Ｎｎ）とに関連づけられ、それぞれの数値には条件が付いている場合もあれば付いていない場合もある。所与の一連のｎ個の数値は、FORK命令に関連するアドレスから始まる命令のｎ個の連続グループを意味する。関連条件が一切付いていない数値は、対応するグループの命令を将来スレッドとして無条件に実行できることを意味する。関連条件が付いている数値は、コンパイル時指定条件が実行時に真と評価された場合のみ、対応するグループの命令の将来スレッドによる実行が有効になるはずであることを意味する。関連数値ｓは、スレッドの先頭にあって、対応するソース・レジスタと宛先レジスタに有効のマークを付けるためにデコード可能なｓ個の命令を意味するが、このスレッドはその命令に関連する動作を実際に実行する必要はない。

【００４８】７．FORK_M_SUSPEND

この命令の命令コードは、将来スレッドの先頭を識別するアドレスと、１組のマスク（Ｍ１、Ｍ２、・・・、Ｍｎ）とに関連づけられ、それぞれのマスクには条件が付いている場合もあれば付いていない場合もある。関連条件が一切付いていないマスクは、将来スレッドの実行用の有効なソース・オペランドを無条件に保持する、１組のアーキテクチャ化レジスタを表している。関連条件が付いているマスクは、コンパイル時指定条件が実行時に真と評価された場合のみ、将来スレッドの実行用の有効なソース・オペランドを保持すると想定することができる、１組のアーキテクチャ化レジスタを意味する。

【００４９】８．FSKIP

この命令の命令コードは、１つのマスクと、数値ｓとに関連づけられている。この命令を検出すると、将来スレッドは、次のｓ個の命令の取出し、デコード、実行をスキップすることができる。さらに将来スレッドは、１組の定義済みレジスタ・セットに有効なオペランドを保持しているというマークを付けるためにこのマスクを使用する。メイン・スレッドはこの命令をＮＯＰとして取り扱う。

【００５０】９．SKPMG

この命令を検出すると、将来スレッドは、次のコンパイル時指定数の命令（通常はスピル・ロード）をデコードし、対応するソース・レジスタと宛先レジスタに有効のマークを付けることができるが、このスレッドはその命令に関連する動作を実際に実行する必要はない。この命令がメイン・スレッドによって検出された場合は、将来スレッドが個のSKPMG命令のアドレスの先頭に事前にフォークされているかどうかを判定するために検査が行われる。フォークされている場合は、メイン・スレッドと対応する将来スレッドという２つのスレッドのマシン状態を適切に組み合わせることにより、メイン・スレッドが対応する将来スレッドと組み合わされ、メイン・スレッドは、将来スレッドが中断された命令に続く命令から実行を再開する。このアドレスへの事前フォークが一切ない場合は、メイン・スレッドは、この命令に続く一連の命令シーケンスを引き続き実行する。このような命令の重要性については後述する。

【００５１】新しい命令の形式の詳細説明：新しい命令の形式を示す図５ないし図１３の詳細説明は以下の通りである。

【００５２】１．FORK <addr_1>, <addr_2>, ..., <add

r_n>
図５のFORK命令（ブロック１０）は、１つの命令コード・フィールド（ブロック１１）と、それぞれが１つの将来スレッドの開始命令アドレスを識別する１つまたは複数のアドレス・フィールドaddr_1、addr_2、・・・、addr_n（ブロック１２－１、１２－２、・・・、１２－ｎ）とを含む。

【００５３】２．UNCOND_SUSPEND
図６のUNCOND_SUSPEND命令（ブロック２０）は、１つの命令コード・フィールドを含む。

【００５４】３．SUSPEND <mode>, <cond_1> <cond_2> ... <cond_n>
図７のSUSPEND命令（ブロック３０）は、SUSPEND命令コード・フィールド（ブロック３１）と、モード・フィールド（ブロック３２）と、条件フィールド（ブロック３３）とを含む。本発明の好ましい実施例では、１つまたは複数の分岐からなるシーケンスの結果に関するコンパイル時推測をcond_1、cond_2、・・・、cond_n（ブロック３３－１、３３－２、・・・、３３－ｎ）としてコード化するために条件フィールドを使用することができる。この特定の条件フィールド・コード化の意味については、以下に詳述する。

【００５５】モード・フィールドは、２通りの方法のうちの１つで条件フィールド内の１組の条件を解釈するために使用する。モード・フィールドが有効（Ｖ）に設定されている場合は、スレッド管理ユニットは、SUSPEND命令に関連する<cond_1>～<cond_n>のうち、コンパイル時指定条件のいずれか１つが実行時に偽と評価された場合に、SUSPEND命令に関連する依存領域内の１組の命令の結果を破棄する。モード・フィールドが無効（Ｉ）に設定されている場合は、スレッド管理ユニットは、SUSPEND命令に関連する<cond_1>～<cond_n>というコンパイル時指定条件のすべてが実行時に真と評価された場合に、SUSPEND命令に関連する依存領域内の１組の命令の結果を破棄する。直観的に言えば、コンパイラは、フォーク点から組合せ点までの優良経路をコード化するために有効モード設定を使用し、フォーク点から組合せ点までの不良経路をコード化するために無効モード設定を使用するはずである。

【００５６】一連の条件のうちの第１の条件cond_1は、SUSPEND命令を含む将来スレッドをフォークしたあとで実行時にフォーキング・スレッドによって検出された第１の固有の条件付き分岐に関連づけられ、一連の条件のうちの第２の条件cond_2は、SUSPEND命令を含む将来スレッドをフォークしたあとで実行時にフォーキング・スレッドによって検出された第２の固有の条件ブランチに関連づけられ、以下同様になる。異なる命令位置に常駐する分岐だけが固有と見なされる。さらに、好ましい実施例では特定の分岐結果のコンパイル時推測をコード化する条件は、実行（Ｔ）、非実行（Ｎ）、無指定（Ｘ）のいずれかにすることができる。あるいは、これらの条件に関連する推測は、実行（Ｔ）または非実行（Ｎ）のいずれかになるように制限することができる。

【００５７】条件コード化形式をさらに明確にするため、以下のコード化例を検討する。

【００５８】○　SUSPEND V, T X N
このコード化は、推測が該当する場合のみ、この条件付き中断命令のあとに続く命令が有効であることを意味する。すなわち、SUSPEND命令に関連する<cond_1>～<cond_n>というコンパイル時指定条件のすべてが実行時に真と評価された場合に、SUSPEND命令に関連する依存領域内の１組の命令の結果である。第１の制御流れ条件では、SUSPEND命令を含むスレッドをフォークしたあとで実行時にフォーキング・スレッドによって検出された第１の固有の条件付き分岐が実行されると想定する。第２のこのような分岐は、どちらに進んでもよい（すなわち、制御非依存分岐）とコンパイラによって認められ、第３のこのような分岐は、非実行であるとコンパイラによって想定されている。

【００５９】○　SUSPEND I, N T X N T X T
このコード化は、推測が該当する場合のみ、この条件付き中断命令のあとに続く命令が無効であることを意味する。すなわち、SUSPEND命令に関連する<cond_1>～<cond_n>というコンパイル時指定条件のすべてが実行時に真と評価された場合にのみ、SUSPEND命令に関連する依存領域内の１組の命令の結果が破棄される。第１の制御流れ条件では、SUSPEND命令を含むスレッドをフォークしたあとで実行時にフォーキング・スレッドによって検出された第１の固有の条件付き分岐が実行されないと想定する。第２のこのような分岐は、実行であるとコンパイラによって想定され、第３のこのような分岐は、どちらに進んでもよい（すなわち、制御非依存分岐）とコンパイラによって認められ、第４のこのような分岐は、非実行であるとコンパイラによって想定され、第５のこのような分岐は、実行であると想定され、第６のこのような分岐は、どちらに進んでもよいと認められ、第７のこのような分岐は、実行であると想定されている。

【００６０】ただし、フォーク後で組合せ前の領域内のフォーキング・スレッド・コードがループなしになるように制限されている場合は、フォーク後にフォーキング・スレッド内で検出される分岐の動的シーケンスがすべて固有のものになるはずであることに留意されたい。すなわち、このような状況では、第１の固有の条件付き分岐は単に第１の動的検出条件付き分岐になり、第２の固有の条件付き分岐は単に第２の動的検出条件付き分岐になり、以下同様になるはずである。

【００６１】上記の条件形式は、FORK_SUSPEND、FORK_S_SUSPEND、FORK_M_SUSPENDの各命令の場合にコンパイル時推測条件を指定する際にも使用する。好ましい実施例では、FORK_SUSPEND、FORK_S_SUSPEND、FORK_M_SUSPEND

の各命令で使用する条件フィールド・コード化において、SUSPEND命令に関連する<cond_1>～<cond_n>のうち、コンパイル時指定条件のいずれか１つが実行時に偽と評価された場合に、スレッド管理ユニットがSUSPEND命令に関連する依存領域内の１組の命令の結果を破棄することを意味する、有効モード・フィールド設定を想定している。

【００６２】４．FORK_SUSPEND <addr>, <N1, cond_1> ... <Nn, cond_n>

図８のFORK_SUSPEND命令（ブロック４０）は、命令コード・フィールド（ブロック４１）と、アドレス・フィールド（ブロック４２）と、それぞれが１つのカウント・フィールドと１つまたは複数の条件とに関連づけられている１つまたは複数の条件フィールド（ブロック４３－１、４３－２、・・・、４３－ｎ）とを含む。条件用の好ましい形式は、有効モード・フィールドを想定し、SUSPEND命令のコンテキストで前述したものと同じである。

【００６３】５．SKIP <n>

図９のSKIP命令（ブロック５０）は、命令コード（ブロック５１）と、SKIP命令のコンテキストで前述したように、この命令以降で実行をスキップすることができる命令の数を指定するカウント・フィールド（ブロック５２）とを含む。

【００６４】６．FORK_S_SUSPEND <addr>, <N>, <N1, cond_1> ... <Nn, cond_n>

図１０のFORK_S_SUSPEND命令（ブロック６０）は、命令コード・フィールド（ブロック６１）と、アドレス・フィールド（ブロック６２）と、（SKIP命令のコンテキストで）前述した意味で実行をスキップすることができる、スレッドの先頭にある命令の数を指定するカウント・フィールド（ブロック６３）と、それぞれが１つのカウント・フィールドと１つまたは複数の条件とに関連づけられている１つまたは複数の条件フィールド（ブロック６４－１、６４－２、・・・、６４－ｎ）とを含む。条件用の好ましい形式は、有効モード・フィールドを想定し、SUSPEND命令のコンテキストで前述したものと同じである。

【００６５】７．FORK_M_SUSPEND <addr>, <M1, cond_1> ... <Mn, cond_n>

図１１のFORK_M_SUSPEND命令（ブロック７０）は、命令コード・フィールド（ブロック７１）と、アドレス・フィールド（ブロック７２）と、それぞれが１つのマスク・フィールドと１つまたは複数の条件とに関連づけられている１つまたは複数の条件フィールド（ブロック７３－１、７３－２、・・・、７３－ｎ）とを含む。それぞれのマスク・フィールドは、有効なソース・オペランドを保持する１組のアーキテクチャ化レジスタを指定するレジスタ・マスクを含むが、関連条件は実行時に適用されるものとする。条件用の好ましい形式は、有効モード・フィールドを想定し、SUSPEND命令のコンテキストで前述したものと同じである。

【００６６】８．FSKIP <mask> <n>

図１２のFSKIP命令（ブロック８０）は、命令コード・フィールド（ブロック８１）と、１組のレジスタを定義するマスク・フィールド（ブロック８２）と、FSKIP命令のコンテキストで前述したように、完全にスキップすることができる命令の数を指定するカウント・フィールド（ブロック８３）とを含む。

【００６７】９．SKPMG <n>

図１３のSKPMG命令（ブロック９０）は、命令コード・フィールド（ブロック９１）と、SKPMG命令のコンテキストで前述したように、この命令以降で実行をスキップすることができる命令の数を指定するカウント・フィールド（ブロック92）とを含む。

【００６８】組合せアクション：フォーク済みスレッドとフォーキング・スレッドとの組合せ：対応するフォーキング・スレッド（たとえば、メイン・スレッド）がフォーク済み（将来）スレッドの先頭に達すると、フォーク済み将来スレッドがフォーキング・スレッドと組み合わされる。この組合せは、フォーク済みスレッドによって定義されたＣＰＵ状態が置き換えられ、残りの状態がフォーキング・スレッドから保持されるように、２つのスレッドのＣＰＵ状態を組み合わせることによって実施される。通常、あるスレッドのＣＰＵ状態は、そのスレッドによって使用され定義されるアーキテクチャ上可視のレジスタを含むはずである。フォーキング・スレッド・プログラム・カウンタは、組み合わされたフォーク済みスレッドによって正しく実行された命令が再実行されず、組み合わされたフォーク済みスレッドによって実行されていない命令が適切に実行されるように実行を続行するために更新される。この場合、正しく実行された命令とは、重要なプログラム依存関係に違反しないような命令を意味する。フォーキング・スレッドは、組み合わされたスレッドの最新実行点を過ぎても実行を続行し、組み合わされた将来スレッドによって正しく実行された命令は、組合せプロセスの終わりに完了の対象となる。組み合わされた将来スレッドに関連する資源は、組合せプロセスの終わりに解放される。ただし、組合せの時点では、フォーク済み将来スレッドがすでに中断されているか、まだ活発に実行中であるかのいずれかであることに留意されたい。いずれも場合も、組合せプロセスが終わると、組み合わされた将来スレッドが効果的に存在を停止する。また、UNCOND_SUSPENDなどの明示的な中断プリミティブがない場合は、組合せが行われるまでフォーク済み将来スレッドがいつも実行を続けるはずであることにも留意されたい。

【００６９】フォークの任意選択性：本発明で提案した命令の新規の特徴は、コンパイル時にそれを使用する場合、実行時ＣＰＵ資源に関するいかなる想定も必要とし

ない点である。実際の実施態様の積極性に応じて、特定のＣＰＵが将来スレッドを実際にフォークできる場合もあれば、できない場合もある。すなわち、ＣＰＵの観点から見ると、FORK命令の検出に応答して実行時に実際にフォークするかどうかは、完全に任意選択である。このような命令のユーザ（たとえば、コンパイラ）は、保留の将来スレッドの数を追跡する必要がなく、また、実行時に確かに従う（すなわち、将来スレッドをフォークする）べき特定のフォークを想定することもできない。

【００７０】コンパイラは、個別の（将来）スレッドとして実行可能な制御およびデータ非依存コード領域を識別する。ただし、コンパイラは、このようなスレッドが並列実行されることを想定するような追加の再構造化または最適化を一切実行しない。たとえば、コンパイラは、挿入されたFORK命令のいずれかが実行時にＣＰＵによって無視されると、正しいプログラム実行の保証が必要になるはずのスピル・コードを保持する。スピル・コードとは、アーキテクチャ上可視のＣＰＵレジスタの内容を命令キャッシュ内の所与の位置に格納し、その後、別の介入格納を行わずに同じ位置の内容を再ロードするためにコンパイル時に挿入される１組の命令を意味する。ただし、スピル・コードの実行は、将来スレッドとしてそれを実行している間は冗長になる可能性があることに留意されたい。将来スレッド実行中のこのようなスピル・コードの処理を最適化するため、本発明はSKIP命令とFSKIPおよびSKPMGなどのその変形態様を追加している。これらは、冗長なスピル・コードの実行を低減または解消するためのコンパイル時のヒントを可能にするものである。この新しい命令の意味の詳細は、前述の通りである。

【００７１】FORK命令の任意選択性の直接的な結果として、それぞれが０個またはそれ以上のスレッドをフォークできる、この強化型マシン・アーキテクチャの各種実施態様に応じて再コンパイルする必要がないことに留意されたい。同様に、新しい命令のいずれも含まない古い２進コードについても、再コンパイルの必要はない。

【００７２】将来スレッドでの複数条件付き中断の解釈：FORK命令に応答してフォークされた将来スレッドが、無条件中断を検出する前に一連の条件付き中断を検出する可能性がある。それぞれの条件付き中断は、いまだに共通のフォーク点に関連してしかも他の条件付き中断とは無関係に解釈されている。したがって、各種の制御流れ推測を将来スレッドの様々な部分に関連づけることは可能である。ここでSUSPEND命令Ａについて検討する。FORK、SUSPEND、UNCOND_SUSPEND、FORK_S_SUSPEND、FORK_M_SUSPEND、またはSKPMG命令以外のいくつかの命令の後で、別のSUSPEND命令ＢがＡの後に続くと想定する。通常、SUSPEND命令Ｂの後にはUNCOND_SUSPEND命令が続くはずである。ここで、SUSPEND命令Ａに関連するコンパイル時条件が実行時に偽になると判定されていると想定する。コンパイルを簡略化し、将来スレッドでの状態維持を低減するために、本発明の好ましい実施例では、破棄の時期を命令ＡとＢとの間に限定するのではなく、命令ＡとUNCOND_SUSPEND命令との間にすべての命令の結果を破棄できるだけである。

【００７３】組合せ点の識別の簡略化：コンパイル時には、将来スレッド内のすべてのスピル・ロードをグループ化し、そのグループを将来スレッドの実行が始まるブロックの先頭に移動させることが可能な場合もある。すべての潜在的将来スレッドの最初の実行が新しいSKPMG命令になるようにさらにコンパイラが保証する場合、この命令は、スキップ可能なスピル・ロードの標識ならびに将来スレッドの開始用のマーカの両方の役割を果たす。この命令の意味については、前述の通りである。ただし、（SKPMGの形での）このような将来スレッド・マーカがない場合、メイン・スレッドは、絶えずその命令アドレスをすべての事前フォーク済み将来スレッドと照らし合わせて検査し、組合せが必要かどうかを検出する必要がある場合もあることに留意されたい。また、スキップされる命令の数がゼロであっても、この解釈では将来スレッド・マーカの追加機能を果たすので、コンパイラはこのSKPMG命令を挿入しなければならない。

【００７４】図３および図４は、１次実行方法（ＰＥＭ）と呼ばれる本発明の実行方法の諸ステップを示す流れ図である。本発明の方法の説明とともに、図３および図４の詳細説明を以下に示す。

【００７５】１．フォーク点の検出（ブロック２１０）：本発明で提案した新しい命令とは無関係に、当技術分野で既知の技法を使用して静的順序の命令シーケンスを生成する。この命令シーケンスを分析して、１組のフォーク点を判定する。フォーク点とは、静的命令シーケンス中の位置であって、使用可能なマシン状態が順次追跡順序で後から（ただし、フォーク点の直後ではない）現れる１組または複数組の命令の並列実行を開始できる位置を意味する。フォーク点の識別には、先行技術で既知の技法を使用して対応するプログラム依存性グラフ（制御依存性グラフとデータ依存性グラフの組合せ）の一部または全部に基づいて行う、データおよび制御依存性分析が含まれる。たとえば、分岐命令を解決すると、実質的に分岐命令に制御依存している命令のスレッド用のフォーク点に到達することができる。

【００７６】２．FORKの挿入（ブロック２２０）：コンパイル時に潜在的フォーク点のうちの０個またはそれ以上に０個またはそれ以上のFORK命令を挿入する。この位置では、FORK命令は、フォーク点に関連する０個またはそれ以上の潜在的将来スレッドの開始アドレスを識別することができる。特定のFORK命令とそのフォーク済み将来スレッド（複数も可）との関連づけがある場合は、その関連づけが前述のＴＭユニットによって管理される。

【００７７】３．静的シーケンスのロード（ブロック２

３０）：固定位置から始まるメモリ・システム（図１のブロック１００）に前のステップ（FORKの挿入、ブロック２２０）後に生成した静的順序の命令シーケンスをロードする。この固定位置では、メモリ・システムと中央演算処理装置の命令キャッシュとのインタフェースが取られ、静的シーケンスの続きが定期的に命令キャッシュに転送される。

【００７８】４．取出しと組合せ検査（ブロック２４０）：現行アドレス以降について、メイン・プログラム・カウンタ（すなわち、メイン・スレッドとして）によりシーケンスをアドレス指定し、プログラム・カウンタを更新することにより、命令キャッシュから命令シーケンスを取り出す。命令キャッシュ内で見つからない命令はメイン・メモリからキャッシュに取り出される。命令の取出しとともに、現行命令取出しアドレス以降について、組み合わされていない１つまたは複数の将来スレッドがあるかどうかを判定するための検査も行われる。この暗黙の組合せ検査の実行は、ＴＭユニット（図１のブロック１３０）も担当する。この検査は通常、組み合わされていない（保留）すべての将来スレッドの開始アドレスとそれぞれの命令取出しアドレスとの比較を含むはずである。

【００７９】５．スレッド妥当性検査（ブロック２５０）：１つまたは複数の将来スレッドが別の実行スレッド（たとえば、メイン・スレッド）の命令取出しアドレスに事前にフォークされていると前のステップ（ブロック２４０）で判定された場合は、１つまたは複数の推測の結果、プログラム依存関係の違反のためにこのような将来スレッドのそれぞれによって実行された命令の一部または全部を破棄する必要があるかどうかを確認するために、追加の検査がＴＭユニットによって行われる。

【００８０】６．組合せ（ブロック２６０）：前のステップ（スレッド妥当性検査、ブロック２５０）で識別されたフォーク済み将来スレッドのうち、有効に実行された部分が、前述の組合せ動作によりメイン・スレッドと組み合わされる。

【００８１】７．デコード（ブロック２７０）：取り出した命令をディスパッチャでデコードする。その命令のうちの１つまたは複数がFORK命令としてデコードされたかどうかを確認するために検査する。

【００８２】８．メイン・スレッドの実行（ブロック２８０）：前のステップ（デコード、ブロック２７０）でいずれかの命令がFORK以外の命令としてデコードされた場合、（図１のブロック１４０を使用して）命令依存関係を分析し、適切な機能ユニット（図２のブロック１８０）上で実行するために（図１のブロック１５０を使用して）それらをスケジューリングすることにより、実行を続行する。

【００８３】９．完了（ブロック２９０）：前述のように、完了ユニット（図２のブロック１９０）により命令実行を完了する。ステップ４～９に記載した取出し、デコード、実行のプロセスは続行される。

【００８４】１０．フォーク能力の判定（ブロック３００）：上記のブロック２７０に関連するステップ（デコード）で命令がFORK命令としてデコードされた場合は、追加の将来スレッドをフォークするために使用可能なマシン資源があるかどうかを判定するための検査が行われる。将来スレッドをフォークするのに必要なマシン資源としては、使用可能なプログラム・カウンタと、スレッド状態を保管するために使用可能な内部バッファ空間などがある。

【００８５】１１．フォーク（ブロック３１０）：使用可能な資源がある場合は、FORK命令に関連するアドレス（複数も可）を将来プログラム・カウンタ（複数も可）にロードすることにより、ＴＭユニットが将来スレッド（複数も可）をフォークする。これにより将来スレッド（複数も可）の実行が始まるが、将来スレッドの開始マシン状態（プログラム・カウンタを除く）は、フォーク点でのメイン・スレッド（関連FORK命令をデコードするスレッド）のものと同じになる。

【００８６】１２．将来スレッドの実行（ブロック３２０）：上記のステップ（４）～（８）と同様に、フォーキング・スレッドの実行と並行して将来スレッドの実行が続行される。ただし、メイン・プログラム・カウンタとメイン・スレッド・ディスパッチャの代わりに、将来プログラムカウンタの１つと将来スレッド・ディスパッチャの１つをそれぞれ使用し、メイン・スレッドをフォーキング・スレッドと呼ぶ。

【００８７】１３．将来スレッドの停止（ブロック３３０）：将来スレッドがフォーキング・スレッドと組み合わされた後、または将来スレッドがＴＭユニットによって破棄された後、将来スレッドの実行が中断され、関連資源が解放される。

【００８８】前述の１次実行方法（ＰＥＭ）の強化例をいくつか以下に説明する。

【００８９】代替実施例１：

１．ＰＥＭのステップ（２）は、以下の追加サブステップを有する。

○　すべての将来スレッドの終わりにUNCOND_SUSPEND命令が挿入される。

【００９０】２．ＰＥＭのステップ（１２）は、以下の追加サブステップを有する。

UNCOND_SUSPEND命令を検出すると、その対応将来スレッドの実行中に将来スレッドは無条件にそのスレッドを中断する。

【００９１】３．ＰＥＭのステップ（８）は、以下の追加サブステップを有する。

○　その対応将来スレッド以外のスレッドが（たとえば、メイン・スレッド内で）実行するためにUNCOND_SUSPEND命令が検出された場合は、その命令は無視される。

【００９２】代替実施例２：
１．代替実施例１を含むＰＥＭのステップ（１）は、以下の追加サブステップを有する。
○　すべてのUNCOND_SUSPEND命令に対応して、対応将来スレッドに０個またはそれ以上のSUSPEND命令を挿入することができ、そこでそれぞれのSUSPEND命令が１つの条件に関連づけられる。
【００９３】２．代替実施例１を含むＰＥＭのステップ（２）は、以下の追加サブステップを有する。
○　SUSPEND命令に関連するコンパイル時指定条件が実行時に真と評価される場合のみ、SUSPEND命令に関連する依存領域内の１組の命令が対応将来スレッドでの実行に有効であると見なされる。したがって、将来スレッドの実行が条件付き中断命令を検出するときまでに関連推測が無効であると分かっている場合には、条件付き中断点で（ＴＭユニットによって）将来スレッドを強制的に中断することもできる。
【００９４】３．代替実施例１を含むＰＥＭのステップ（３）は、以下の追加サブステップを有する。
○　その対応将来スレッド以外のスレッドが（たとえば、メイン・スレッド内で）実行するためにSUSPEND命令が検出された場合は、その命令は無視される。
【００９５】代替実施例３：
１．代替実施例２を含むＰＥＭのステップ（１）は、以下の追加サブステップを有する。
○　０個またはそれ以上のSKIP命令を将来スレッドに挿入することができ、そこで、それぞれのSKIP命令が数値ｓに関連づけられる。
【００９６】２．代替実施例２を含むＰＥＭのステップ（２）は、以下の追加サブステップを有する。
○　その対応将来スレッドの実行中に関連数値ｓを含むSKIP命令を検出すると、この命令に続く次のｓ個の命令だけをデコードする必要が生じ、これらの命令の残りの実行はスキップすることができる。これらの命令で使用するソース・レジスタと宛先レジスタは、有効オペランドを保持するものとしてマークを付けることができるが、いずれかの機能ユニット上で実行するためにこれらのｓ個の命令をスケジューリングする必要はない。
【００９７】３．代替実施例２を含むＰＥＭのステップ（３）は、以下の追加サブステップを有する。
○　その対応将来スレッド以外のスレッドが（たとえば、メイン・スレッド内で）実行するためにSKIP命令が検出された場合は、その命令は無視される。
【００９８】代替実施例４：
１．代替実施例２を含むＰＥＭのステップ（１）は、以下の追加サブステップを有する。
○　０個またはそれ以上のFSKIP命令を将来スレッドに挿入することができ、そこで、それぞれのFSKIP命令が、１組のアーキテクチャ化レジスタを定義するマスクと、数値ｓとに関連づけられる。
【００９９】２．代替実施例２を含むＰＥＭのステップ（２）は、以下の追加サブステップを有する。
○　その対応将来スレッドの実行中にマスクと数値ｓを含むFSKIP命令を検出すると、この命令に続く次のｓ個の命令をスキップすることができる。すなわち、これらの命令は、取出し、デコード、または実行を行う必要がない。マスクで識別されたレジスタは、有効オペランドを保持するものとしてマークを付けることができる。
【０１００】３．代替実施例２を含むＰＥＭのステップ（３）は、以下の追加サブステップを有する。
○　その対応将来スレッド以外のスレッドが（たとえば、メイン・スレッド内で）実行するためにFSKIP命令が検出された場合は、その命令は無視される。
【０１０１】代替実施例５：
１．代替実施例２を含むＰＥＭのステップ（１）は、以下の追加サブステップを有する。
○　すべての将来スレッドの先頭に１つのSKPMG命令が挿入され、そこで、それぞれのSKPMG命令が数値ｓに関連づけられる。
【０１０２】２．代替実施例２を含むＰＥＭのステップ（２）は、以下の追加サブステップを有する。
○　その対応将来スレッドの実行中に関連数値ｓを含むSKPMG命令を検出すると、この命令に続く次のｓ個の命令だけをデコードする必要が生じ、これらの命令の残りの実行はスキップすることができる。これらの命令で使用するソース・レジスタと宛先レジスタは、有効オペランドを保持するものとしてマークを付けることができるが、いずれかの機能ユニット上で実行するためにこれらのｓ個の命令をスケジューリングする必要はない。
【０１０３】３．代替実施例２を含むＰＥＭのステップ（３）は、以下の追加サブステップを有する。
○　その対応将来スレッド以外のスレッドが（たとえば、メイン・スレッド内で）実行するためにSKPMG命令が検出された場合は、SKPMG命令の命令アドレス以降について、将来スレッドがすでにフォークされているかどうかを判定するための組合せ検査が行われる。
【０１０４】４．ＰＥＭのステップ（４）の暗黙の組合せ検査はここでは不要であり、そのため除去される。
【０１０５】代替実施例６：
１．ＰＥＭのFORKの挿入ステップ（すなわち、ステップ３）は、以下のステップに置き換えられる。
○　０個またはそれ以上の潜在的フォーク点に０個またはそれ以上のFORK_SUSPEND命令を挿入する。その場合、FORK_SUSPEND命令は、関連の潜在的将来スレッドの開始アドレスを識別するアドレスと、一連の数値とを含み、それぞれの数値には条件が付いている場合もあれば付いていない場合もある。所与の一連の数値は、FORK_SUSPEND命令に関連するアドレスから始まる命令の連続グループを意味する。特定のFORK_SUSPEND命令とそのフォーク済み将来スレッドとの関連づけがある場合は、前述のＴ

Mユニットによってその関連づけが管理される。

【０１０６】２．ＰＥＭのフォーク能力の判定ステップ（すなわち、ステップ１０）は、以下のステップに置き換えられる。

○　命令がFORK_SUSPEND命令としてデコードされた場合は、追加の将来スレッドをフォークするために使用可能なマシン資源があるかどうかを判定するための検査が行われる。

【０１０７】３．ＰＥＭのフォーク・ステップ（すなわち、ステップ１１）は、以下のステップに置き換えられる。

○　使用可能な資源がある場合は、FORK_SUSPEND命令に関連するアドレス（複数も可）を将来プログラム・カウンタ（複数も可）にロードすることにより、将来スレッドをフォークする。

【０１０８】４．ＰＥＭの将来スレッドの実行ステップ（すなわち、ステップ１２）は、以下の追加サブステップを有する。

○　FORK_SUSPEND命令に関連する数値列は、以下のように対応将来スレッドの実行を制御する。数値たとえばｎに関連条件が一切付いていない場合は、ｎ個の命令からなる対応グループを将来スレッドとして無条件に実行できることを意味し、数値たとえばｍに関連条件が付いている場合は、コンパイル時指定条件が実行時に真と評価される場合のみ、ｍ個の命令からなる対応グループの将来スレッド実行が有効になるはずである。

【０１０９】代替実施例７：

１．ＰＥＭのFORKの挿入ステップ（すなわち、ステップ３）は、以下のステップに置き換えられる。

○　０個またはそれ以上の潜在的フォーク点に０個またはそれ以上のFORK_S_SUSPEND命令を挿入する。その場合、FORK_S_SUSPEND命令は、関連の潜在的将来スレッドの開始アドレスを識別するアドレスと、数値たとえばｓと、一連の数値とを含み、それぞれの数値には条件が付いている場合もあれば付いていない場合もある。所与の一連の数値は、FORK_S_SUSPEND命令に関連するアドレスから始まる命令の連続グループを意味する。

【０１１０】２．ＰＥＭのフォーク能力の判定ステップ（すなわち、ステップ１０）は、以下のステップに置き換えられる。

○　命令がFORK_S_SUSPEND命令としてデコードされた場合は、追加の将来スレッドをフォークするために使用可能なマシン資源があるかどうかを判定するための検査が行われる。

【０１１１】３．ＰＥＭのフォーク・ステップ（すなわち、ステップ１１）は、以下のステップに置き換えられる。

○　使用可能な資源がある場合は、FORK_S_SUSPEND命令に関連するアドレス（複数も可）を将来プログラム・カウンタ（複数も可）にロードすることにより、将来スレッドをフォークする。

【０１１２】４．ＰＥＭの将来スレッドの実行ステップ（すなわち、ステップ１２）は、以下の追加サブステップを有する。

○　FORK_S_SUSPEND命令に関連する数値列は、以下のように対応将来スレッドの実行を制御する。対応スレッドを将来スレッドとして実行している間は最初のｓ個の命令だけをデコードすることができ、これらの命令で使用するソース・レジスタと宛先レジスタは、有効オペランドを保持するものとしてマークを付けることができるが、いずれかの機能ユニット上で実行するためにこれらのｓ個の命令をスケジューリングする必要はない。さらに、数値たとえばｎに関連条件が一切付いていない場合は、ｎ個の命令からなる対応グループを将来スレッドとして無条件に実行できることを意味し、数値たとえばｍに関連条件が付いている場合は、コンパイル時指定条件が実行時に真と評価される場合のみ、ｍ個の命令からなる対応グループの将来スレッド実行が有効になるはずである。

【０１１３】代替実施例８：

１．ＰＥＭのFORKの挿入ステップ（すなわち、ステップ３）は、以下のステップに置き換えられる。

○　０個またはそれ以上の潜在的フォーク点に０個またはそれ以上のFORK_M_SUSPEND命令を挿入する。その場合、FORK_M_SUSPEND命令は、関連の潜在的将来スレッドの開始アドレスを識別するアドレスと、１組のマスクとを含み、それぞれのマスクには条件が付いている場合もあれば付いていない場合もある。

【０１１４】２．ＰＥＭのフォーク能力の判定ステップ（すなわち、ステップ１０）は、以下のステップに置き換えられる。

○　命令がFORK_M_SUSPEND命令としてデコードされた場合は、追加の将来スレッドをフォークするために使用可能なマシン資源があるかどうかを判定するための検査が行われる。

【０１１５】３．ＰＥＭのフォーク・ステップ（すなわち、ステップ１１）は、以下のステップに置き換えられる。

○　使用可能な資源がある場合は、FORK_M_SUSPEND命令に関連するアドレス（複数も可）を将来プログラム・カウンタ（複数も可）にロードすることにより、将来スレッドをフォークする。

【０１１６】４．ＰＥＭの将来スレッドの実行ステップ（すなわち、ステップ１２）は、以下の追加サブステップを有する。

○　FORK_M_SUSPEND命令に関連するマスク列は、以下のように対応将来スレッドの実行を制御する。対応スレッドを将来スレッドとして実行している間、FORK_M_SUSPENDに関連し、条件が一切付いていないマスクは、将来スレッドの実行のための有効ソース・オペランドを無条件

に保持する１組のアーキテクチャ化レジスタを表し、条件に関連するマスクは、コンパイル時指定条件が実行時に真と評価される場合のみ、将来スレッドの実行のための有効ソース・オペランドを保持すると想定することができる１組のアーキテクチャ化レジスタを意味する。命令のソース・レジスタ・オペランドに関連するコンパイル時指定条件が実行時に真に該当しない場合は、ＴＭユニットが将来スレッド内の命令の一部または全部の結果を破棄する。

【０１１７】代替実施例９：

１．ＰＥＭのメイン・スレッドの実行ステップ（すなわち、ステップ８）は、以下の追加サブステップを有する。

○　スレッド実行中のすべての分岐解決（すなわち、条件付き分岐を実行するかどうかの判定と、関連目標アドレス）はＴＭユニットに連絡される。ＴＭユニットはこの情報を使用して、間違った分岐アドレスにフォークされた将来スレッドと依存スレッドを破棄する必要があるかどうかを判定する。これにより、後述するように命令の制御依存ブロックの同時実行が可能になる。

【０１１８】代替実施例１０：

１．ＰＥＭの取出しと組合せ検査ステップ（すなわち、ステップ４）は、以下の追加サブステップを有する。

○　事前にフォークされたスレッドのいずれかが既定のタイムアウト期間より長い間、組み合わされずにいたかどうかを確認する検査を含むように、組合せ検査が拡張される。このようなスレッドはすべてＴＭユニットによって破棄される。

【０１１９】新しい命令のコード化の詳細説明：図１４ないし図１７は、新しい命令の一部の好ましいコード化例を示している。ビット位置０は最上位ビット位置を意味し、ビット位置３１は最下位ビット位置を意味する。

【０１２０】１．FORK（図１４）

この命令（ブロック１１１）では、ビット０～５を使用して１次命令コード４を使用している。将来スレッドの開始アドレスの相対アドレスはビット位置６～２９の２４ビットのアドレス・フィールドにコード化される。最後の２ビットであるビット位置３０と３１は、FORK命令の代替形式のコード化を行うための拡張命令コード・フィールドとして使用される。この２ビットは、FORK命令のこのバージョンでは０に設定される。

【０１２１】２．UNCOND_SUSPEND（図１５）

この命令（ブロック２２２）では、ビット位置０～５の１次命令コード１９を使用している。拡張命令コード・フィールドのビット２１～３０は、それを同じ１次命令コードを含む他の命令と区別するために５１４に設定されている。ビット３１は、この無条件中断命令を条件付き中断（SUSPEND）命令と区別するために０に設定されている。

【０１２２】３．SUSPEND（図１６）

この命令（ブロック３３３）では、ビット位置０～５の１次命令コード１９を使用している。拡張命令コード・フィールドのビット２１～３０は、それを同じ１次命令コードを含む他の命令と区別するために５１４に設定されている。ビット３１は、この条件付き中断命令を無条件中断（UNCOND_SUSPEND）命令と区別するために１に設定されている。コンパイル時分岐推測は、実行、非実行、無指定のいずれかになる。したがって、ビット位置７～２０を使用して７つのコンパイル時分岐推測Ｃ１～Ｃ７のそれぞれについて、２ビットずつ使用する。このシーケンスの第１の条件Ｃ１（ビット７と８）は、SUSPEND命令を含む将来スレッドのフォーク後に実行時にフォーキング・スレッドによって検出される第１の固有の条件付き分岐に関連づけられ、このシーケンスの第７の条件Ｃ７は、SUSPEND命令を含む将来スレッドのフォーク後に実行時にフォーキング・スレッドによって検出される第７の固有の条件付き分岐に関連づけられている。モード・フィールドはビット位置６にコード化される。このコード化に関連する意味については、SUSPEND命令のコンテキストで既に前述した通りである。

【０１２３】４．FORK_SUSPEND（図１７）

この命令（ブロック４４４）でも、ビット位置０～５で上記のFORK命令に使用したものと同じ１次命令コード４を使用している。しかし、拡張命令コード・フィールド（ビット３０と３１）は、それをFORK命令と区別するために１に設定されている。将来スレッドの開始アドレスの相対アドレスはビット位置２０～２９の１０ビットのアドレス・フィールドにコード化される。コンパイル時分岐推測は、実行、非実行、無指定のいずれかになる。したがって、４つのコンパイル時分岐推測Ｃ１～Ｃ４のそれぞれについて、２ビットずつ使用する。このシーケンスの第１の条件Ｃ１は、SUSPEND命令を含む将来スレッドのフォーク後に実行時にフォーキング・スレッドによって検出される第１の固有の条件付き分岐に関連づけられ、このシーケンスの第４の条件Ｃ４は、SUSPEND命令を含む将来スレッドのフォーク後に実行時にフォーキング・スレッドによって検出される第４の固有の条件付き分岐に関連づけられている。第１の数値Ｎ１（ビット６～８）は、Ｃ１（ビット９と１０）とＣ２（ビット１１と１２）の両方に関連する条件が実行時に真に該当すると評価されると想定して、将来スレッドの開始アドレスから始まる有効な命令の数を意味する。これに対して、Ｎ２（ビット１３～１５）は、Ｃ３（ビット１６と１７）とＣ４（ビット１８と１９）の両方に関連する条件が実行時に真に該当すると評価されると想定して、将来スレッドの開始アドレスから始まる有効な命令＋Ｎ１個の命令の数を意味する。

【０１２４】例

図１８ないし図２０は、本発明で提案した命令の一部をコード・シーケンスのサンプルで使用した場合を示して

いる。図示のコード・シーケンスは、任意で分岐命令で終わる、非分岐命令のブロックに分割されている。使用する命令ニーモニックは、本発明で導入したもの（たとえば、FORK）か、またはＰｏｗｅｒＰＣアーキテクチャ（ＰｏｗｅｒＰＣはＩＢＭの商標である）のものかのいずれかである。条件付き分岐で終わるコードシーケンスのブロックは、分岐を実行しない場合に制御の転送先になるブロックに続く辺Ｎと、分岐を実行する場合に制御の転送先になるブロックに続くもう１つの辺Ｔとを有する。

【０１２５】図１８および図１９は、複数の命令からなる制御非依存ブロック間で推測するために本発明で提案した命令を使用する場合を示している。図１８および図１９のＢ１とＢ１２のような様々な制御非依存ブロックの同時取出し、デコード、推測、実行を可能にするために、FORK、SUSPEND、UNCOND_SUSPENDの各命令が使用されている。制御がブロックＢ０からＢ１に達すると、ブロックＢ１のFORK命令を使用して、制御非依存ブロックＢ１２とＢ１との並行実行を開始する。ただし、Ｂ１を実行するメイン・スレッドはいくつかの経路の１つをたどることができるが、それらの経路はいずれも将来スレッドとして実行されるブロックＢ１２に到達することに留意されたい。同様に、ブロックＢ１の終わりにある分岐の解決によってＢ３に至る場合には、制御非依存Ｂ９の並行実行のためにFORK命令が使用される。ブロックＢ３を実行するスレッドは、ブロックＢ６またはＢ７の実行後にＢ９から始まる将来スレッドと組み合わされる。

【０１２６】アーキテクチャ化レジスタ２とメモリ位置ｍｅｍ６それぞれの更新の結果発生する重要な依存関係を観察するために、ブロックＢ９とＢ１２を実行する将来スレッドで無条件中断すなわちUNCOND_SUSPEND命令が使用される。ブロックＢ３の終わりにある分岐の結果として、フォーキング・スレッド（ブロックＢ３を実行するもの）が実行時にブロックＢ７に移行し、ブロックＢ６（レジスタ３を更新するもの）を回避すると想定して、次の２つの命令を推論実行するために、ブロックＢ９では条件付き中断すなわちSUSPEND命令が使用される。同様に、制御がブロックＢ１０（レジスタ４を更新するもの）に移行しないと想定して、次の４つの命令を推論実行するためにSUSPEND命令が使用される。ただし、回避される経路、すなわちブロックＢ２とＢ１０を介してブロックＢ１のフォーク点からブロックＢ１２の組合せ点に至る経路は、経路表現ＴＸＴを使用してコンパイル時にコード化されることに留意されたい。この表現は、フォーク点後の第１の固有の条件付き分岐すなわちＢ１の終わりにある分岐が実行され、第２の分岐すなわちＢ２の終わりにある分岐はどちらにも進むことができ、Ｂ８の終わりにある分岐も実行されることを意味する。ただし、この場合、複数の優良経路（すなわち、レジスタ４を一切更新しない経路）が存在することに留意されたい。ブロックＢ２の終わりにある分岐はブロックＢ４またはブロックＢ５のいずれかに進むことができ、Ｂ８の終わりにある分岐が実行されず、Ｂ１１に至る場合はこれらの経路のいずれも優良と見なされるはずである。

【０１２７】フォークの任意選択性を保証するために、図１９のブロックＢ１２の先頭にあるスピル・ロードがコンパイラによって保存されていることに留意されたい。また、Ｂ１２が将来スレッドとして実行される場合は、スピル・ロードの冗長実行を最適化するために図１９でSKIP命令が使用されることにも留意されたい。

【０１２８】図２０は、複数の命令からなる制御依存ブロック間で推測するためにFORK命令とSUSPEND命令を使用する場合を示している。FORK命令は、ブロックＢ１００から制御依存ブロックＢ２００とＢ３００にフォークするために使用されている。制御依存ブロックＢ２００とＢ３００は、推論実行され、並列である。メイン・スレッドはブロックＢ１００を実行するが、フォーク済み将来スレッドはブロックＢ２００とブロックＢ３００を実行する。ブロックＢ１００の終わりにある分岐を解決すると、ＴＭユニットは間違った分岐結果に関して条件付けされた将来スレッドを破棄する。たとえば、分岐を実行する場合、Ｂ２００から始まる将来スレッドが破棄される。

【０１２９】潜在的利点

この項では、上記で提案した命令が前述の諸問題を解決するのにどのように役立つかを詳しく説明する。

【０１３０】１．命令取出しのボトルネックの緩和

上記の例に示したように、提案したフォーク命令と中断命令は、現行スーパースカラー・プロセッサの命令取出しのボトルネックに対処する新規の方法を提供する。コンパイラは、これらの命令を使用して、任意の距離にある（動的に）制御非依存ブロックを指し示すことができる。制御非依存とは、プログラム制御がフォーク点に達した場合に、これらの将来ブロックに達するように拘束される（当然のことながら、予測できないような流れの変更を行うことができる割込みは一切ないものと想定する）ことを意味する。したがって、そのブロックの制御依存性が解決されるとただちに（制御の流れを待たずに）命令を取り出すことができる。また、その制御依存性が得られる分岐（制御の流れが得られる分岐ではない）が間違って予測された場合のみ、推論方式で取り出した命令を破棄しなければならない。たとえば、ブロックＢ９の命令は、ブロックＢ１でのその共用制御依存性が解決されるか、推測された直後に、ブロックＢ３の命令とともに取り出すことができる。さらに、ブロックＢ９からの命令は、ブロックＢ３の終わりにある分岐が間違って予測された場合ではなく、ブロックＢ１の終わりにある制御依存分岐が間違って予測された場合のみ、無駄な取出しと見なすか、破棄しなければならない。制御

依存性の概念を持たない従来のスーパースカラーは、ブロックＢ７とＢ９がブロックＢ３の終わりにある分岐の従来の制御流れ推測により取り出され、その後、これが予測謝りであると判明した場合、ブロックＢ７（またはＢ６）ならびにＢ９のその推論取出しを破棄するはずである。

【０１３１】２．制御非依存ブロック間でのデータ非依存性の活用

これらのブロックに至る可能性のあるすべての制御流れ経路とはデータ非依存でもある制御非依存ブロックの命令は、このような制御非依存ブロックへの複数フォークを介して、同時かつ非推論方式で実行することができる。たとえば、ブロックＢ９（Ｂ３とは制御非依存である）の最初の３つの命令は、ブロックＢ３、Ｂ６、Ｂ７（Ｂ３からＢ９への１組の制御流れ経路上の１組の基本ブロック）の命令とはデータ非依存である。したがって、これらの命令は、提案したフォーク命令と中断命令を使用して、非推論方式で取り出して実行することができる。

【０１３２】３．制御非依存ブロック間でのデータ依存性の推測

将来スレッドの活動とメイン・スレッドの活動とのオーバラップを高めるためには、将来スレッド内の潜在的データ依存性に関する何らかの形式の推測が必要である。図１８および図１９の例を検討する。この場合、ブロックＢ１～Ｂ１１にはレジスタ４の定義が１つしかない。これはブロックＢ１０で定義されている。メイン・スレッドの制御の流れについて推測する、すなわち、メイン・スレッドの制御の流れがブロックＢ１０に達しないと想定すると、ブロックＢ１２の先頭から始まる将来スレッドとブロックＢ１に継続するメイン・スレッドとのオーバラップを高めることが可能である。ブロックＢ１０の違反命令に至る正確な制御の流れは、提案した条件付き中断命令の一部として、＜ＴＸＴ＞としてコード化される。ただし、制御流れの推測はコンパイル時に行われ、このため、静的分岐予測（またはプロファイル主導あるいはその両方）の技法だけに基づいて行われることに留意されたい。また、この場合の正味効果は、条件付き中断命令と無条件中断命令との間の命令を推論方式で格上げすることと同様であることにも留意されたい。しかし、それぞれの保護された（格上げされた）命令の一部として制御流れ条件をコード化する、保護された（または格上げされた）命令の既知の技法とは異なり、提案した技法では、条件付き中断命令と無条件中断命令とを使用して１群の命令用の条件をコード化する。この手法の重要な利点としては以下のものがある。

【０１３３】○　アーキテクチャ上の影響が小さいこと

前に示したように、提案した方式の主な利点は、そのアーキテクチャ上の影響が相対的に最小限であることである。フォーク命令と中断命令（そのうち、フォーク命令だけが１次命令コード空間を必要とする）の追加を除き、既存の命令コード化は影響を受けない。したがって、格上げ手法とは異なり、提案した方式では、制御流れの推測をコード化するために格上げしたそれぞれの命令の命令コードで使用可能なビット数に依存しない。

【０１３４】○　推測した制御流れのコード化の正確さ

新しい（中断）命令では制御流れの推測が排他的にコード化されるので、より多くのビット数を使用してそれを正確にコード化することができる。たとえば、格上げ方式では、想定した流れ経路に沿って格上げした命令の深さをコード化するためにのみ、ある妥協点に達しなければならなかった（それぞれの分岐は想定した結果ビットを有し、最も可能性の高い追跡経路を示していた）。この妥協点は、格上げしたそれぞれの命令の命令コードに収容できるように、推測した制御流れを簡潔にコード化するために必要であった。この妥協点の結果、制御非依存分岐の予測を誤ると、推論方式で実行し格上げした命令が不必要に破棄されてしまった。本明細書で提案した手法では、推測した制御流れ経路に沿った制御非依存分岐が、ＮまたはＴではなくＸによって正しくコード化される。このため、将来スレッドで推論方式で実行された命令は、制御非依存分岐の予測を誤っても破棄されない。

【０１３５】○　小規模なコード拡大

典型的なパーコレーションおよび格上げ技法では、想定した追跡から外れる経路でのコード・コピーまたはパッチアップ・コードが必要になる場合が多い。これは、コード・サイズの大幅な拡大に至る可能性がある。提案した技法はこれらのオーバヘッドのいずれも持たないので、唯一のコード拡大は、１組の命令によって共用されるフォーク命令と中断命令によるものである。

【０１３６】○　順次例外処理の実施の単純化

提案した技法には上方コード・モーションが一切なく、推論方式で実行したコードは依然としてその元の位置だけにとどまっている。したがって、メイン・スレッドが命令の原因である例外を含む将来スレッドと組み合わされるまで、例外処理を容易に遅延させることができる。すなわち、例外を引き起こす可能性のある推論命令の元の位置に明示的にマークを付ける必要もなく、正しい順序で例外を処理することができる。

【０１３７】○　正確な割込みの実施の単純化

この提案の固有のメイン・スレッドは、順次プログラム順序で完了する最後の命令をいつも正確に把握している。したがって、割込みを正確に処理するために重要な余分なハードウェアを設ける必要は全くない。

【０１３８】４．コンパイルとマシン実施態様との結合解除

前述のように、フォークの任意選択性のため、提案したアーキテクチャのコンパイルは、多数の活動スレッドが可能なマシンを想定して行うことができることに留意さ

れたい。しかも、実際のマシン実施態様は、使用可能なマシン資源に応じて、このようなフォークのほとんどまたはこのようなフォークの一部に従うか、このようなフォークのいずれにも従わないかを選択することができる。したがって、このコンテキストの大部分では、マシン実施態様からコンパイルを結合解除することができる。これは、少数または多数の活動スレッドが可能なマシンについて個別に再コンパイルする必要がないことも意味する。

【０１３９】５．ループ反復の並列実行

提案したフォーク命令と中断命令は、ネストしたループでの反復並列性を効率よく活用するためにも使用することができる。たとえば、SPECint92ベンチマークの１つから前述したサンプル・ループについて検討する。このループの内側ループ反復は、前の反復に対して制御とデータの両面で依存している。しかし、内側ループのそれぞれの活動化（すなわち、外側ループ反復）は前のものとは無関係である。このため、前のものが完了するまで待たずに、しかも内部ループの何らかの制御およびデータ非依存反復の予測を誤ったときに外側ループ反復から実行した命令を不必要に破棄せずに、マシンが内側ループの活動化を何度も開始できるようにするため、提案したフォーク命令（外側ループ・ボディを開始する）をコンパイラが使用することは可能である。

【０１４０】６．レジスタ・プレッシャの緩和

互いにデータ非依存でもある制御非依存基本ブロック内の命令は、取り出すことも実行することもできない。起こりうる明らかな疑問は、このようなデータおよび制御非依存命令が、同一基本ブロック内にまとめて入れるのに十分なほどパーコレーションされていないのかということである。優良コンパイラはこのようなパーコレーションを実施しようと最善を尽くすはずであるが、必ずこれらの命令をまとめてグループ化できるわけではない。前述のように、すべてのデータおよび制御非依存命令を効率よくまとめてグループ化できるようにするには、コンパイラは、適切なコード化を行うのに十分なアーキテクチャ化レジスタを有する必要がある。たとえば、図１８および図１９で使用する例の推測上のマシンが４つのアーキテクチャ化レジスタ、すなわちレジスタ１～レジスタ４しか提供しないものと想定する。このようなマシン用のコンパイラは、追加のスピル・コードを挿入せずに、制御およびデータ非依存命令を基本ブロックＢ１およびＢ１２に単純にグループ化することはできない。フォーク機構により、コンパイラは追加のスピル・コードがなくても基礎となるデータ非依存性を伝えることができる。事実、既存のスピル・コードの一部は、Ｂ１２が実行時に実際にフォークされると冗長（たとえば、基本ブロックＢ１２内の最初の２つのロード）になる可能性がある。このようなスピル・コードは、前述のようにSKIP命令を使用して最適化することができる。

【０１４１】７．制御依存ブロック間の推測

これまでの説明でフォークを使用したのは、制御非依存ブロックの並列実行の場合だけであった。この概念は、制御依存ブロックを含むようにさらに拡張することができる。さらにこれは、両方の分岐経路を推論方式で実行できる能力を意味する。追加の実施コストが必要ではあるが、このような推測はいずれもアーキテクチャに対してこれ以上影響を及ぼすことはない。この形式の推測の追加の有用性は、ある程度は（１つの分岐経路に沿って）現行推論スーパースカラーにすでに使用されているが、さらに検討する必要がある。図２０で使用する例は、どちらもＢ１００に対して制御依存しているブロックＢ２００およびＢ３００のような制御依存ブロック間で推測するためにフォーク命令と中断命令を使用する場合を示している。ブロックＢ１００のフォークも、両方の分岐経路に沿って推測し、実行時の実際の制御の流れ（Ｂ２００またはＢ３００）に基づいて適切に命令を破棄するためのものである。

【０１４２】８．スレッド管理の簡略化

○　スレッド間の同期：固有のメイン・スレッドと残りの将来スレッドという概念は、スレッド間の同期を簡略化した機構を提供し、低いオーバヘッドを意味する。明示的な中断点では、将来スレッドは単にそれ自体を中断し、メイン・スレッドの制御が将来スレッドに到達するのを待つだけである。あるいは、その実行中の様々な時点で将来スレッドが他のスレッドとの明示的スレッド間同期を試みることもできる。しかし、このようにスレッド間同期がさらに精巧になると、ハードウェア／ソフトウェアのオーバヘッドが増すことを意味する。

【０１４３】○　スレッド間通信：アーキテクチャ化マシン状態のコピーによるフォークと、前述の組合せ動作の概念により、オーバヘッドの低いスレッド間通信の機構が提供される。オーバヘッドがこれよりかなり高い代替機構は、たとえば、メッセージにより、活動スレッド間の連続通信プロトコルを提供する明示通信プリミティブをもたらすことができる。

【０１４４】○　スレッドのスケジューリング：　本発明で提案した機構では、結果的に（前述のように）フォークの任意選択性が得られるが、FORK命令に応答してスレッドをスケジューリング（フォーク）するために実行時スレッド・スケジューリング・ハードウェアを必要としないので、動的スレッド・スケジューリングも簡略化される。したがって、スレッド・スケジューリング・ハードウェアは、すべてのFORK命令が暗示する将来スレッド（複数も可）の待ち行列化と管理を負担する必要がない。動的スレッド・スケジューリングのオーバヘッドがこのように低下しているので、再コンパイルせずに各種のマシン実施態様に適合できることなど、その他の利点により、静的スレッド・スケジューリングに比べ、それがより魅力的なものになる。

【０１４５】まとめとして、本発明の構成に関して以下の事項を開示する。

【０１４６】（１）コンピュータ内の中央演算処理装置において、

ａ．複数の命令を有する命令キャッシュ・メモリであって、その命令キャッシュが１つまたは複数の命令キャッシュ・ポートをさらに有する命令キャッシュ・メモリと、

ｂ．複数のプログラム・カウンタからなるプログラム・カウンタ・バンクであって、それぞれのプログラム・カウンタが命令キャッシュ内の１つまたは複数の命令を独立してアドレス指定し、アドレス指定した命令を命令キャッシュ・ポートの１つにポーティングすることができる、プログラム・カウンタ・バンクと、

ｃ．複数のディスパッチャからなり、それぞれのディスパッチャが命令バッファを１つずつ有し、それぞれのディスパッチャが１つまたは複数の命令キャッシュ・ポートから命令を受け取り、受け取った命令をその命令バッファに入れ、命令をデコードし、その関連バッファ内の命令間の依存関係を分析することができる、ディスパッチャ・バンクと、

ｄ．それぞれのスレッドがプログラム・カウンタの１つを使用して実行した一連の命令シーケンスを有する、１つまたは複数のスレッドをフォークし、０または１つ以上のスレッド間通信を処理するスレッド管理ユニットと、

ｅ．すべてのディスパッチャから命令を受け取り、１つまたは複数の機能ユニット上で実行するために命令をスケジューリングするスケジューラと、

ｆ．すべてのスレッド内の命令によってアクセス可能な１つまたは複数のアーキテクチャ化レジスタからなる固定セットを含むレジスタ・ファイルとを含み、それにより、１つまたは複数の命令スレッドが機能ユニットによって並列に実行されることを特徴とする、中央演算処理装置。

（２）プログラム・カウンタ・バンク内のプログラム・カウンタの１つがメイン・スレッド内の命令を追跡し、メイン・スレッドが順次追跡順序で最も早いスレッドであることを特徴とする、上記（１）に記載の装置。

（３）１つまたは複数のディスパッチャが、そのディスパッチャが分析中に解決できない１つまたは複数の依存関係について推測し、スレッド管理ユニットが、１つまたは複数の推測の結果として、プログラム依存関係の違反のために将来スレッドのいずれかによって実行された１つまたは複数の命令を破棄する必要があるかどうかを判定することができ、スレッド管理ユニットがこのような違反命令を破棄することを特徴とする、上記（２）に記載の装置。

（４）スレッド管理ユニットは、FORK命令が検出されたときに指定のアドレスから始まる将来スレッドをフォークすることができ、FORK命令がコンパイル時に命令スレッドに挿入され、FORK命令が１つまたは複数の将来スレッドの先頭を識別することを特徴とする、上記（３）に記載の装置。

（５）FORK命令が１つの命令コード・フィールドと１つまたは複数のアドレス・フィールドとを含み、それぞれのアドレスが将来スレッドの開始位置を識別することを特徴とする、上記（４）に記載の装置。

（６）FORK命令の命令コード・フィールドがビット０～５を含み、アドレス・フィールドがビット６～２９を含み、拡張命令コード・フィールドがビット３０と３１とを含むことを特徴とする、上記（５）に記載の装置。

（７）スレッド管理ユニットは、FORK命令が検出されたときに指定のアドレスから始まる将来スレッドをフォークすることができ、UNCOND_SUSPEND命令が検出されたときにその将来スレッドを無条件に中断し、FORK命令とUNCOND_SUSPEND命令がコンパイル時に挿入されることを特徴とする、上記（３）に記載の装置。

（８）FORK命令が１つの命令コード・フィールドと１つまたは複数のアドレス・フィールドとを含み、それぞれのアドレスが将来スレッドの開始位置を識別し、UNCOND_SUSPEND命令が１つの命令コード・フィールドを含むことを特徴とする、上記（７）に記載の装置。

（９）FORK命令の命令コード・フィールドがビット０～５を含み、アドレス・フィールドがビット６～２９を含み、拡張命令コード・フィールドがビット３０と３１とを含み、UNCOND_SUSPENDの命令コードがビット０～５を含む１つの１次命令コード・フィールドと、ビット２１～３１を含む１つの拡張命令コード・フィールドとを有することを特徴とする、上記（８）に記載の装置。

（１０）１つまたは複数のSUSPEND命令を有し、将来スレッドの１つの実行中にSUSPEND命令が検出され、SUSPEND命令に関連するコンパイル時指定条件が実行時に偽と評価された場合にスレッド管理ユニットがSUSPEND命令に関連する依存性領域内の１組の命令の結果を破棄し、SUSPEND命令がコンパイル時に挿入されることを特徴とする、上記（７）に記載に装置。

（１１）SUSPEND命令が、１つのSUSPEND命令コード・フィールドと、１つのモードビットと、１つの条件フィールドとを含むことを特徴とする、上記（１０）に記載の装置。

（１２）SUSPEND命令コードが、ビット０～５を含む１つの１次命令コード・フィールドと、ビット６を占有する１つのモード・フィールドと、ビット６～２０を占有し、それぞれが２ビットの長さの７つの条件サブフィールドから構成される１つの条件フィールドと、ビット２１～３１を含む１つの拡張命令コード・フィールドとを有することを特徴とする、上記（１１）に記載の装置。

（１３）スレッド管理ユニットは、FORK_SUSPEND命令が検出されたときに指定のアドレスから始まる将来スレッ

ドをフォークすることができ、FORK_SUSPEND命令がコンパイル時に命令スレッドに挿入され、FORK_SUSPEND命令が１組または複数組の命令を識別することができ、それぞれの組の命令がそれぞれの組の命令の有効実行を判定する関連条件を任意で有することを特徴とする、上記（３）に記載の装置。

（１４）FORK_SUSPEND命令が、１つの命令コード・フィールドと、１つのアドレス・フィールドと、１つまたは複数の条件フィールドとを含み、それぞれの条件フィールドが１つのカウント・フィールドと１つまたは複数の条件とを有することを特徴とする、上記（１３）に記載の装置。

（１５）FORK_SUSPEND命令が、ビット０～５を含む１つの命令コードと、ビット６～８を含む第１のカウント・フィールドと、第１のカウント・フィールドに関連し、ビット９～１０および１１～１２をそれぞれ含む２つの条件とを有する第１の条件フィールドと、ビット１３～１５を含む第２のカウント・フィールドと、第２のカウント・フィールドに関連し、ビット１６～１７および１８～１９をそれぞれ含む２つの条件とを有する第２の条件フィールドと、ビット２０～２９を含む１つのアドレス・フィールドと、ビット３０および３１を含む１つの拡張命令コード・フィールドとを有することを特徴とする、上記（１４）に記載の装置。

（１６）SKIP命令を検出したときに、将来スレッドが、SKIP命令によって指定された複数の命令をデコードし、実行を行わずに識別された命令の実行を引き受けることを特徴とする、上記（１０）に記載の装置。

（１７）SKIP命令が１つの命令コード・フィールドと１つのカウント・フィールドとを含むことを特徴とする、上記（１６）に記載の装置。

（１８）スレッド管理ユニットは、FORK_S_SUSPEND命令が検出されたときに指定のアドレスから始まる将来スレッドをフォークすることができ、FORK_S_SUSPEND命令がコンパイル時に命令スレッドに挿入され、FORK_S_SUSPEND命令が１組または複数組の命令を識別することができ、それぞれの組の命令がそれぞれの組の命令の有効実行を判定する関連条件を任意で有し、スレッドに開始時に複数の命令を識別するスキップ・カウント・フィールドをさらに有し、実行を行わずに識別された命令の実行を引き受けることを特徴とする、上記（３）に記載の装置。

（１９）FORK_S_SUSPEND命令が、１つの命令コード・フィールドと、１つのアドレス・フィールドと、１つのスキップ・カウント・フィールドと、１つまたは複数の条件フィールドとを含み、それぞれの条件フィールドが１つのカウント・フィールドと１つまたは複数の条件とを有することを特徴とする、上記（１８）に記載の装置。

（２０）スレッド管理ユニットは、FORK_M_SUSPEND命令が検出されたときに指定のアドレスから始まる将来スレッドをフォークすることができ、FORK_M_SUSPEND命令がコンパイル時に命令スレッドに挿入され、FORK_M_SUSPEND命令が１組のレジスタ・マスクを識別することができ、マスクに関連する条件がある場合にその条件が実行時に該当するのであれば、それぞれのマスクが有効ソース・オペランドを保持する複数のアーキテクチャ化レジスタからなるサブセットを識別することを特徴とする、上記（３）に記載の装置。

（２１）FORK_M_SUSPEND命令が、１つの命令コード・フィールドと、１つのアドレス・フィールドと、１つまたは複数の条件フィールドとを含み、それぞれの条件フィールドが１つのレジスタ・マスクと１つまたは複数の条件とを有することを特徴とする、上記（２０）に記載の装置。

（２２）FSKIP命令を検出したときに、将来スレッド・ディスパッチャが取出しと、その結果、この命令に続く指定の数の命令の実行をスキップし、FSKIP命令が有効オペランドを保持する１組のアーキテクチャ化レジスタを指定するレジスタ・マスクを識別することができ、メイン・スレッド・ディスパッチャがこれをＮＯＰとして扱い、FSKIP命令がコンパイル時に命令スレッドに挿入されることを特徴とする、上記（１０）に記載の装置。

（２３）FSKIP命令が、１つの命令コード・フィールドと、１つのマスク・フィールドと、１つのカウント・フィールドとを含むことを特徴とする、上記（２２）に記載の装置。

（２４）SKPMG命令を検出したときに、将来スレッドが、SKPMG命令によって指定された複数の命令をデコードし、実行を行わずに識別された命令の実行を引き受け、メイン・スレッド・ディスパッチャがこの命令を潜在的将来スレッドの開始アドレス用のマーカとして扱い、SKPMG命令がコンパイル時に命令スレッドに挿入されることを特徴とする、上記（１０）に記載の装置。

（２５）SKPMG命令が１つの命令コード・フィールドと１つのカウント・フィールドとを含むことを特徴とする、上記（２４）に記載の装置。

（２６）スレッド管理ユニットが任意でフォークすることができることを特徴とする、上記（１）に記載の装置。

（２７）命令キャッシュがメイン・メモリで置き換えられることを特徴とする、上記（１）に記載の装置。

（２８）中央演算処理装置を備えたコンピュータ・システム上で命令を実行する方法において、（ａ）複数の命令からなる静的シーケンスをコンパイル時に生成し、その命令の静的シーケンスを分析して１組のフォーク点を判定するステップと、（ｂ）コンパイル時に０個またはそれ以上のFORK命令を０個またはそれ以上のフォーク点に挿入するステップと、（ｃ）メモリ内の固定位置から始まるメイン・メモリにその命令の静的シーケンスをロードし、その静的シーケンスのサブシーケンスを命令キ

ャッシュに転送するステップと、（ｄ）現行アドレスから始まるメイン・プログラム・カウンタによりそのシーケンスをアドレス指定することにより、命令キャッシュから命令シーケンスをメイン・スレッドとして取り出し、現行アドレスから始まり、まだ組み合わされていない１つまたは複数の将来スレッドがあるかどうかを判定するために検査するステップと、（ｅ）組み合わされていない将来スレッドの妥当性を検査するステップと、（ｆ）０個またはそれ以上の組み合わされていない将来スレッドの有効に実行された部分をメイン・スレッドに組み合わせるステップと、（ｇ）取り出した命令をディスパッチャでデコードし、１つまたは複数の命令がFORK命令としてデコードされたかどうかを確認するために検査するステップと、（ｈ）命令がFORK命令以外の命令としてデコードされた場合に、命令依存関係を分析し、適切な機能ユニット上で実行するために命令をスケジューリングすることにより、メイン・スレッドを実行するステップと、（ｉ）完了ユニットにより命令実行を完了し、ステップ（ｄ）からこのステップまでを繰り返すステップと、（ｊ）命令がFORK命令としてデコードされた場合に、追加の将来スレッドをフォークするために使用可能なマシン資源があるかどうかを判定するために検査するステップと、（ｋ）使用可能資源がある場合に、FORK命令に関連するアドレスを将来プログラム・カウンタにロードすることにより、将来スレッドをフォークするステップと、（ｌ）それぞれメイン・プログラム・カウンタとメイン・スレッド・ディスパッチャの代わりに将来プログラム・カウンタの１つと将来スレッド・ディスパッチャの１つを使用することによってステップ（ｄ）～（ｈ）を実行することにより、フォーキング・スレッドの実行と並列して将来スレッドを実行し、将来スレッドがメイン・スレッドと組み合わされるか、または将来スレッドがスレッド管理ユニットによって消去された場合に、将来スレッドの実行を中断するステップとを含むことを特徴とする方法。

（２９）前記ステップ（ｂ）が、（ｂ．１）　すべての将来スレッドの終わりにUNCOND_SUSPEND命令を挿入するという追加のサブステップを有し、前記ステップ（ｌ）が、（ｌ．１）　UNCOND_SUSPEND命令を検出したときに将来スレッドの実行を中断するという追加のサブステップを有し、前記ステップ（ｈ）が、（ｈ．１）　その対応将来スレッド以外のスレッドによって実行するために検出された場合に、UNCOND_SUSPEND命令をＮＯＰとして扱うという追加のサブステップを有することを特徴とする、上記（２８）に記載の方法。

（３０）前記ステップ（ｂ）が、（ｂ．２）　すべてのUNCOND_SUSPEND命令に対応する０個またはそれ以上のSUSPEND命令を挿入するという追加のサブステップを有し、前記ステップ（ｌ）が、（ｌ．２）　SUSPEND命令に関連するコンパイル時指定条件が実行時に偽と評価された場合に、SUSPEND命令に関連する依存性領域内の１組の命令を破棄するという追加のサブステップを有し、前記ステップ（ｈ）が、（ｈ．２）　その対応将来スレッド以外のスレッドによって実行するために検出された場合に、SUSPEND命令をＮＯＰとして扱うという追加のサブステップを有することを特徴とする、上記（２９）に記載の方法。

（３１）前記ステップ（ｂ）が、（ｂ．３）　０個またはそれ以上のSKIP命令を将来スレッドに挿入するという追加のサブステップを有し、前記ステップ（ｌ）が、（ｌ．３）　SKIP命令に続く指定の数の命令をデコードし、将来スレッドとしての実行中に、実行を行わずにその指定の数の命令の実行を引き受けるという追加のサブステップを有し、前記ステップ（ｈ）が、（ｈ．３）　その対応将来スレッド以外のスレッドによって実行するために検出された場合に、SKIP命令をＮＯＰとして扱うという追加のサブステップを有することを特徴とする、上記（３０）に記載の方法。

（３２）前記ステップ（ｂ）が、（ｂ．４）　０個またはそれ以上のFSKIP命令を将来スレッドに挿入するという追加のサブステップを有し、前記ステップ（ｌ）が、（ｌ．４）　将来スレッドとしての実行中にFSKIP命令に続く指定の数の命令の取出しをスキップし、関連マスクで識別されたレジスタに有効オペランドを保持するものとしてマークを付けるという追加のサブステップを有し、前記ステップ（ｈ）が、（ｈ．４）　その対応将来スレッド以外のスレッドによって実行するために検出された場合に、FSKIP命令をＮＯＰとして扱うという追加のサブステップを有することを特徴とする、上記（３０）に記載の方法。

（３３）前記ステップ（ｂ）が、（ｂ．５）　すべての将来スレッドの先頭にSKPMG命令を挿入するという追加のサブステップを有し、前記ステップ（ｌ）が、（ｌ．５）　SKIP命令に続く指定の数の命令をデコードし、将来スレッドとしての実行中に、実行を行わずにその指定の数の命令の実行を引き受けるという追加のサブステップを有し、前記ステップ（ｈ）が、（ｈ．５）　その対応将来スレッド以外のスレッドによって実行するためにSKPMGが検出された場合に、SKPMG命令の命令アドレス以降について、過去に将来スレッドがフォークされたかどうかを判定するために検査するという追加のサブステップを有し、前記ステップ（ｄ）が、（ｄ．１）　メイン・プログラム・カウンタによりそのシーケンスをアドレス指定することにより、命令キャッシュから命令シーケンスを取り出すというステップで置き換えられることを特徴とする、上記（３０）に記載の方法。

（３４）前記ステップ（ｂ）が、（ｂ．６）　０個またはそれ以上のFORK_SUSPEND命令を０個またはそれ以上の潜在的フォーク点に挿入するというステップで置き換えられ、前記ステップ（ｊ）が、（ｊ．１）　命令がFORK

_SUSPEND命令としてデコードされた場合に、追加の将来スレッドをフォークするために使用可能なマシン資源があるかどうかを判定するために検査するというステップで置き換えられ、前記ステップ（ｋ）が、（ｋ．１）使用可能資源がある場合に、FORK_SUSPEND命令に関連するアドレス（複数も可）を将来プログラム・カウンタ（複数も可）にロードすることにより、将来スレッドをフォークするというステップで置き換えられ、前記ステップ（ｌ）が、（ｌ．６）　関連するコンパイル時指定条件が実行時に真に該当しない場合に、将来スレッド内の命令の一部または全部の結果を破棄するという追加のサブステップを有することを特徴とする、上記（２８）に記載の方法。

（３５）前記ステップ（ｂ）が、（ｂ．７）　０個またはそれ以上のFORK_S_SUSPEND命令を０個またはそれ以上の潜在的フォーク点に挿入するというステップで置き換えられ、前記ステップ（ｊ）が、（ｊ．２）　命令がFORK_S_SUSPEND命令としてデコードされた場合に、追加の将来スレッドをフォークするために使用可能なマシン資源があるかどうかを判定するために検査するというステップで置き換えられ、前記ステップ（ｋ）が、（ｋ．２）　使用可能資源がある場合に、FORK_S_SUSPEND命令に関連するアドレス（複数も可）を将来プログラム・カウンタ（複数も可）にロードすることにより、将来スレッドをフォークするというステップで置き換えられ、前記ステップ（ｌ）が、（ｌ．７）　将来スレッドの先頭にある指定の数の命令をデコードし、これらの命令の実行を行わずに指定の数の命令の実行を引き受け、関連するコンパイル時指定条件が実行時に真に該当しない場合に、将来スレッド内の命令の一部または全部の結果を破棄するという追加のサブステップを有することを特徴とする、上記（２８）に記載の方法。

（３６）前記ステップ（ｂ）が、（ｂ．８）　０個またはそれ以上のFORK_M_SUSPEND命令を０個またはそれ以上の潜在的フォーク点に挿入するというステップで置き換えられ、前記ステップ（ｊ）が、（ｊ．３）　命令がFORK_M_SUSPEND命令としてデコードされた場合に、追加の将来スレッドをフォークするために使用可能なマシン資源があるかどうかを判定するために検査するというステップで置き換えられ、前記ステップ（ｋ）が、（ｋ．３）　使用可能資源がある場合に、FORK_M_SUSPEND命令に関連するアドレス（複数も可）を将来プログラム・カウンタ（複数も可）にロードすることにより、将来スレッドをフォークするというステップで置き換えられ、前記ステップ（ｌ）が、（ｌ．８）　命令のソース・レジスタ・オペランドに関連するコンパイル時指定条件が実行時に真に該当しない場合に、将来スレッド内の命令の一部または全部の結果を破棄するという追加のサブステップを有することを特徴とする、上記（２８）に記載の方法。

（３７）前記ステップ（ｈ）が、（ｈ．６）　スレッド実行中のすべての分岐解決をＴＭユニットに連絡し、ＴＭユニットがこの情報を使用して、間違った分岐アドレスにフォークされた将来スレッドとすべての依存スレッドを破棄する必要があるかどうかを判定するという追加のサブステップを有することを特徴とする、上記（２８）に記載の方法。

（３８）前記ステップ（ｄ）が、（ｄ．２）　ＴＭユニットが、事前フォーク済みスレッドのいずれかが既定のタイムアウト期間より長い間、組み合わされない状態を維持していたかどうかを判定するために検査し、このようなスレッドを破棄するという追加のサブステップを有することを特徴とする、上記（２８）に記載の方法。

【図面の簡単な説明】

【図１】本発明の方法を実行すると思われる典型的なプロセッサ編成のハードウェアのブロック図である。

【図２】本発明の方法を実行すると思われる典型的なプロセッサ編成のハードウェアのブロック図である。

【図３】本発明の方法の諸ステップを示す流れ図である。

【図４】本発明の方法の諸ステップを示す流れ図である。

【図５】FORK命令の形式構造を示すブロック図である。

【図６】UNCOND_SUSPEND命令の形式構造を示すブロック図である。

【図７】SUSPEND命令の形式構造を示すブロック図である。

【図８】FORK_SUSPEND命令の形式構造を示すブロック図である。

【図９】SKIP命令の形式構造を示すブロック図である。

【図１０】FORK_S_SUSPEND命令の形式構造を示すブロック図である。

【図１１】FORK_M_SUSPEND命令の形式構造を示すブロック図である。

【図１２】FSKIP命令の形式構造を示すブロック図である。

【図１３】SKPMG命令の形式構造を示すブロック図である。

【図１４】FORK、UNCOND_SUSPEND、SUSPEND、FORK_SUSPENDの各命令の形式構造のコード化の好ましい実施例を示す１組のブロック図である。

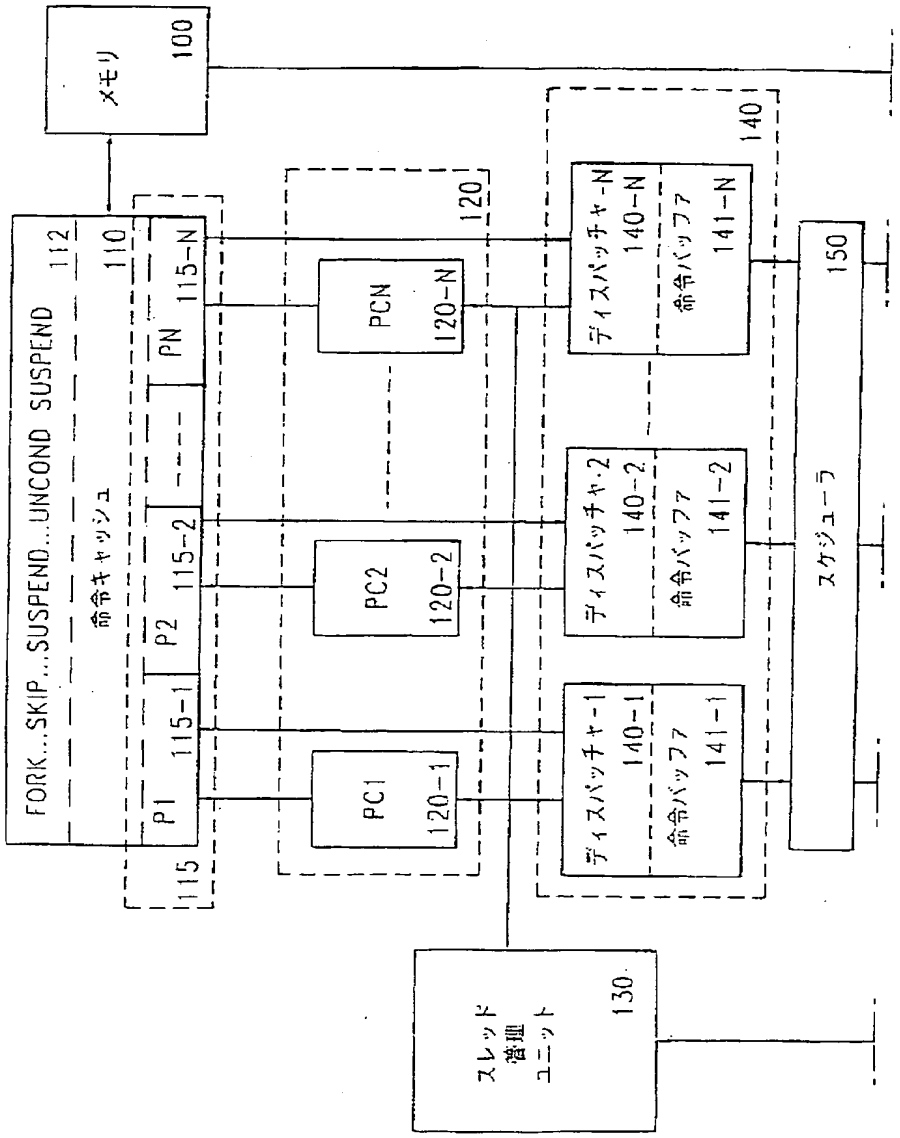
【図１５】FORK、UNCOND_SUSPEND、SUSPEND、FORK_SUSPENDの各命令の形式構造のコード化の好ましい実施例を示す１組のブロック図である。

【図１６】FORK、UNCOND_SUSPEND、SUSPEND、FORK_SUSPENDの各命令の形式構造のコード化の好ましい実施例を示す１組のブロック図である。

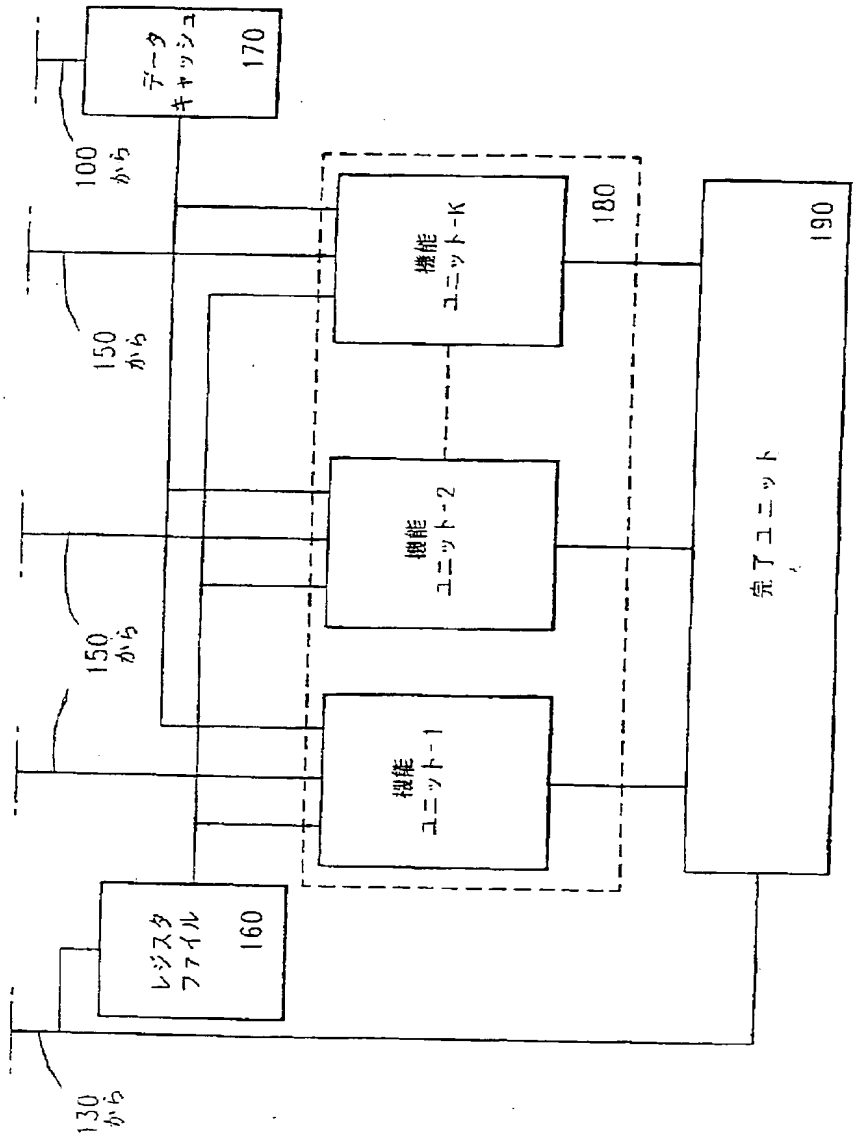
【図１７】FORK、UNCOND_SUSPEND、SUSPEND、FORK_SUSPENDの各命令の形式構造のコード化の好ましい実施例を示す１組のブロック図である。

【図１８】本発明で提案した命令の一部をアセンブリ・コード・サンプルで使用した場合を示す図である。
【図１９】本発明で提案した命令の一部をアセンブリ・コード・サンプルで使用した場合を示す図である。
【図２０】本発明で提案した命令の一部をアセンブリ・コード・サンプルで使用した場合を示す図である。
【符号の説明】
１００　メモリ
１１０　命令キャッシュ
１１５　ポート
１２０　プログラム・カウンタ
１３０　スレッド管理ユニット
１４０　ディスパッチャ
１４１　命令バッファ
１５０　スケジューラ
１６０　レジスタ・ファイル
１７０　データ・キャッシュ
１９０　完了ユニット

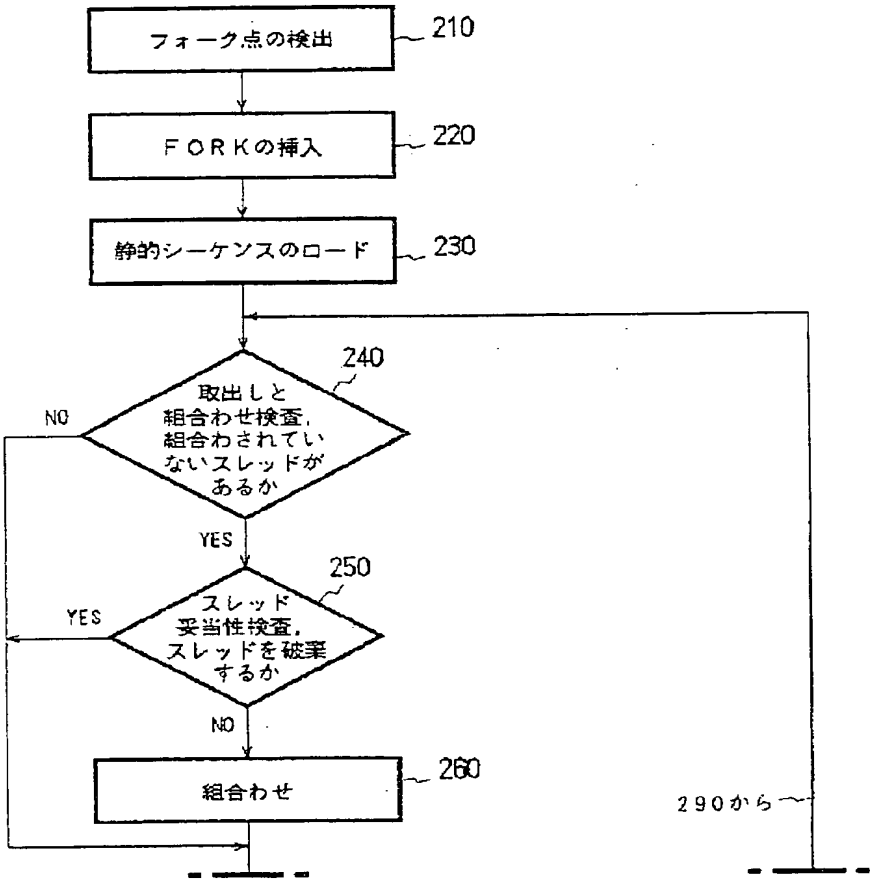
【図１】



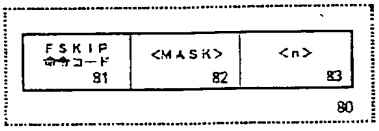
【図２】


データ
キャッシュ
170
100
から
機能
ユニット-K
180
190
150
から
機能
ユニット-2
完了ユニット
150
から
機能
ユニット-1
レジスタ
ファイル
160
130
から

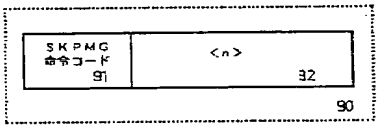
【図３】



【図１２】

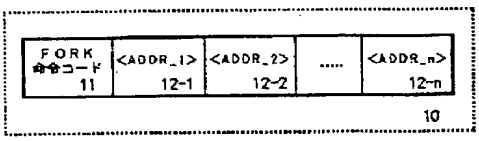


【図１３】

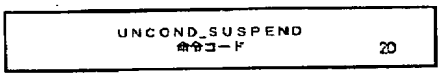


【図５】

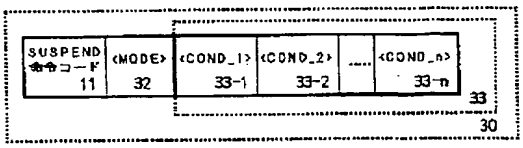
【図６】

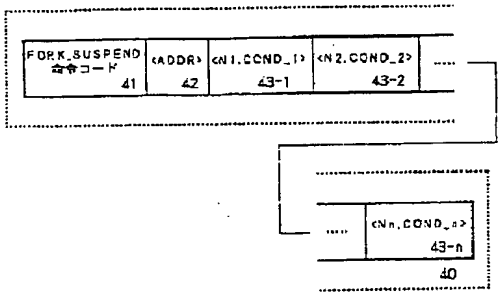


【図７】

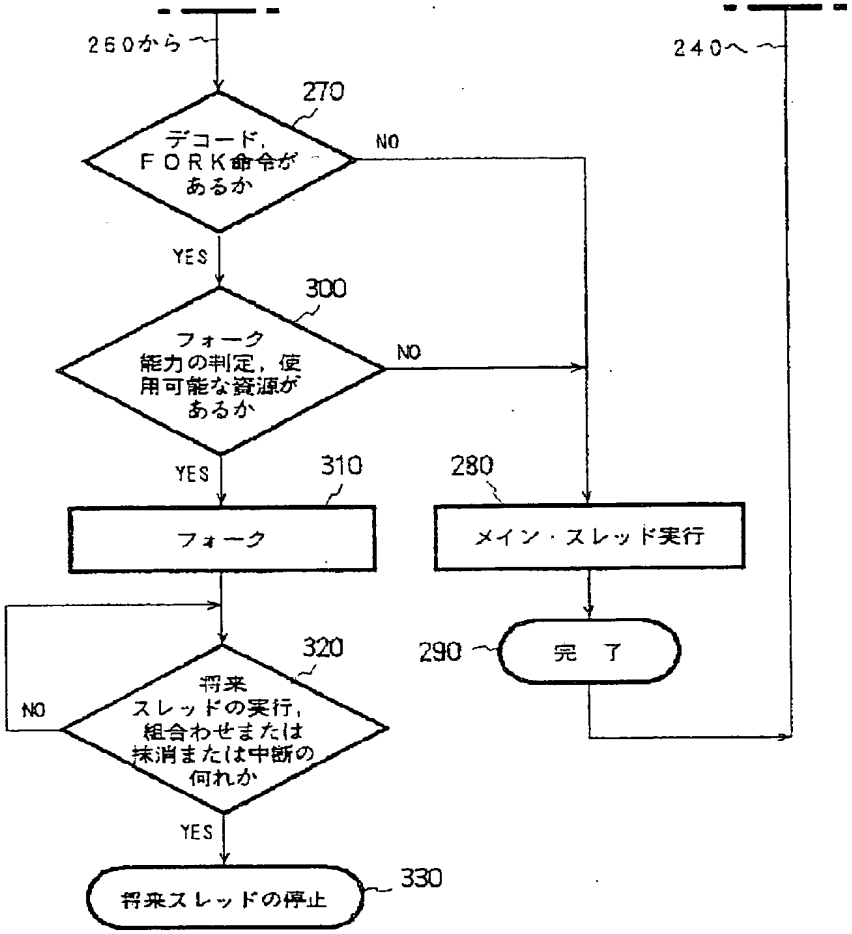


【図８】

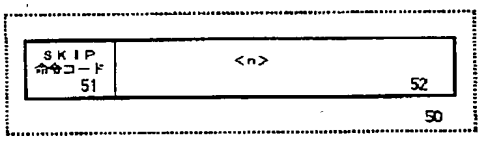




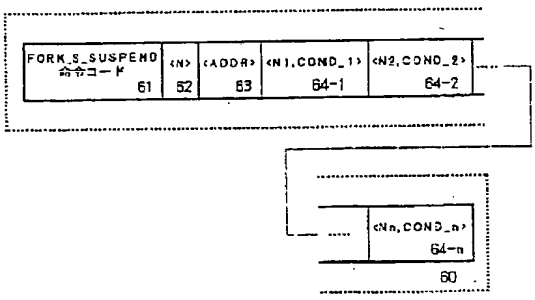
【図４】


250から
270
デコード，FORK命令があるか
NO
YES
300
フォーク能力の判定，使用可能な資源があるか
NO
YES
310
フォーク
280
メイン・スレッド実行
240へ
290
完　了
320
将来スレッドの実行，組合わせまたは抹消または中断の何れか
NO
YES
330
将来スレッドの停止

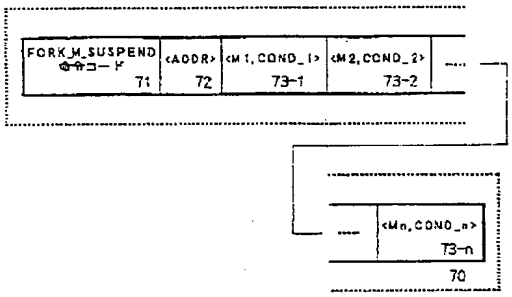
【図９】


SKIP命令コード
51
<n>
52
50

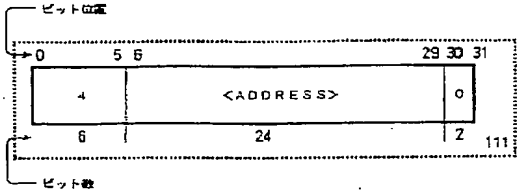
【図１０】


FORK_S_SUSPEND命令コード
61
<N>
52
<ADDR>
63
<N1,COND_1>
64-1
<N2,COND_2>
64-2
<Nn,COND_n>
64-n
60

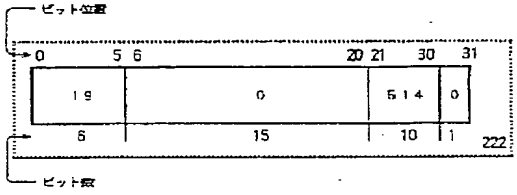
【図１１】



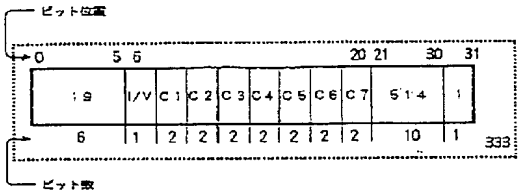
【図１４】



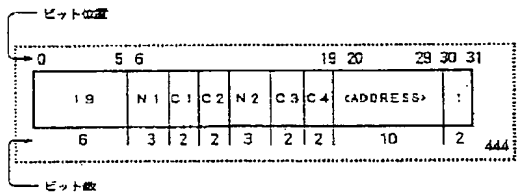
【図１５】



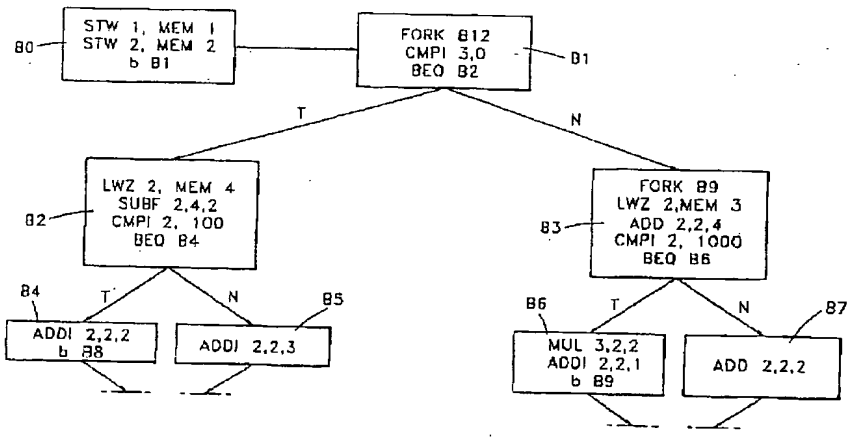
【図１６】



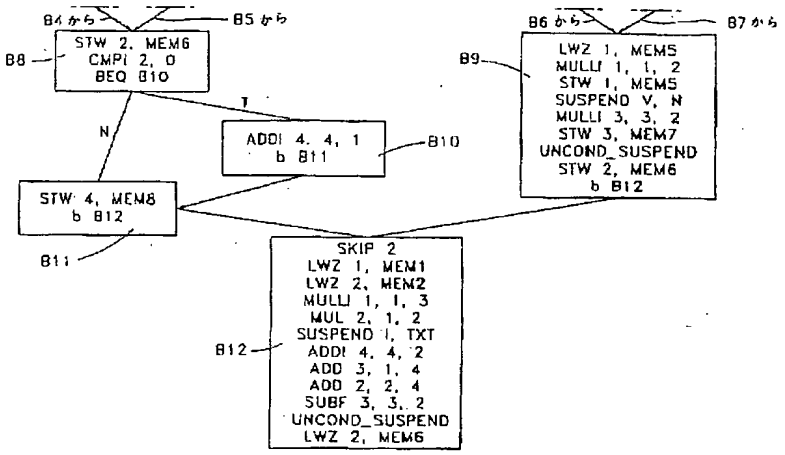
【図１７】



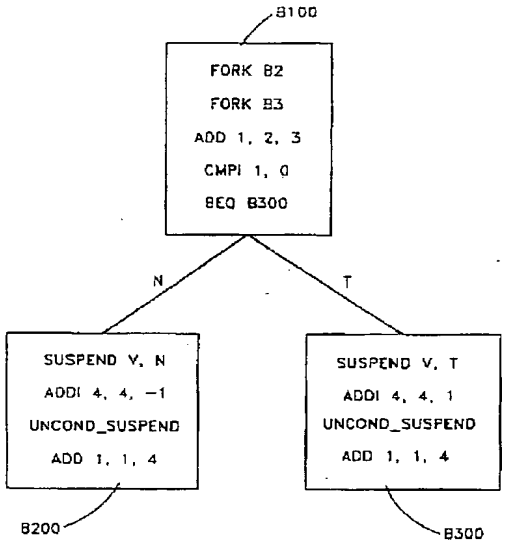
【図１８】



【図１９】



【図２０】



フロントページの続き

(72)発明者　チャールズ・マーシャル・バートン
アメリカ合衆国07641　ニュージャージー州ハワース　プロスペクト・アベニュー　28

(72)発明者　チャオ＝メイ・チュアン
アメリカ合衆国95014　カリフォルニア州キュパーティノ　レインボウ・ドライブ　7585

(72)発明者　リン・フエ・ラム
アメリカ合衆国10598　ニューヨーク州ヨークタウン・ハイツ　ブラックベリー・レーン　770

(72)発明者　ジョン・ケヴィン・オブライエン
アメリカ合衆国10590　ニューヨーク州サウス・サーレム　サーレム・ヒル・ロード　ピー・オー・ボックス　370

(72)発明者　キャスリン・メアリー・オブライエン
アメリカ合衆国10590　ニューヨーク州サウス・サーレム　サーレム・ヒル・ロード　ピー・オー・ボックス　370